MICRODOT, A 4-BIT SYNCHRONOUS MICROCONTROLLER FOR SPACE APPLICATIONS

THESIS

Kirby Michael Watson
First Lieutenant, USAF

AFIT/GE/ENG/01M-20

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/01M-20

MICRODOT, A 4-BIT SYNCHRONOUS MICROCONTROLLER FOR SPACE APPLICATIONS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Kirby Michael Watson, B.S.E.E.

First Lieutenant, USAF

March 2001

AFIT/GE/ENG/01M-20

MICRODOT, A 4-BIT SYNCHRONOUS MICROCONTROLLER FOR SPACE APPLICATIONS

Kirby Michael Watson, B.S.E.E.
First Lieutenant, USAF

Approved:

[illegible] 5 Mar 01

Major Charles P. Brothers, Jr., Ph.D. Date
Thesis Advisor

[illegible] 05MAR01

Lieutenant Colonel James A. Lott, Ph.D. Date
Committee Member

[illegible] 5 Mar 01

Lieutenant Colonel Michael A. Marciniak, Ph.D. Date
Committee Member

Acknowledgements

I would like to thank my parents for having instilled in me the values of hard work and discipline. If not for them, none of this would have been possible. I must thank my loving wife, who has put up with the hectic schedule and hours that comes with an Electrical Engineering degree from AFIT. She kept me going when times were tough and when the end never seemed reachable. I owe her a lot for these last 18 months.

Thank you to all my family who have been patient and understanding throughout my time here at AFIT. I dedicate this thesis to my Grandpa Lentino, who passed away during my time at AFIT. His spirit and love always kept me going strong throughout difficulty and he will always be in my heart.

Thank you Major Charles Brothers for helping me in areas that I knew nothing about. Thanks also go to Lieutenant Colonel James Lott and Lieutenant Colonel Michael Marciniak for being on my thesis committee. Their experience helped me guarantee that everything in this thesis is accurate and makes sense.

Thank you to Dr. Greg Donohoe and the Air Force Research Laboratory (AFRL/VSSE) at Kirtland AFB, NM for giving me this opportunity and funding my test chip. I learned invaluable lessons from this experience and I hope my research will help future AFRL research development programs.

Thank you to Captain Dan Harvala for keeping me sane during the long hours in the VLSI lab. I must thank Greg Richardson for maintaining the VLSI network and continuously increasing my memory space for this work.

Table of Contents

List of Figures

List of Tables

AFIT/GE/ENG/01M-20

Abstract

Satellites have limited power budgets due to the amount of power collected by the satellite's solar panels. The goal is to have a wide range of functionality, while running off a limited power source. Large microprocessors use large amounts of power to report back temperature and chemical sensor data to ground stations. By using small microcontrollers to perform the data collection and minimizing the usage of the larger microprocessors, the satellites will save power. A prototype design of the Microdot 4-bit microcontroller for space applications is presented. Requirements for the Microdot, such as microwatt power consumption and 23 different instructions, are based on research completed at AFRL/VSSE, Air Force Research Laboratory at Kirtland AFB, NM. A brief history of 4-bit microcontrollers and microprocessors, the synchronous design methodologies used, and space-based integrated circuit issues are presented. Various CAD tools were used, implementing both standard cell and full custom logic into the design. The prototype Microdot was fabricated at TSMC using MOSIS to validate the design implementation.

Results from high fidelity simulations indicate the Microdot design has a power consumption of 16.3 mW operating at 1 kHz and consumes 22 mW when operating at the maximum operating clock frequency of 20 MHz. These results indicate that the Microdot can be implemented into space-based systems, while exhibiting low power usage.

MICRODOT - A 4-BIT SYNCHRONOUS MICROCONTROLLER FOR SPACE APPLICATIONS

1. Introduction

1.1. Introduction

The purpose of this research is to investigate, design, and implement a synchronous Very Large Scale Integrated (VLSI) circuit for space applications to relay sensor information from individual sensors back to a command data link within a satellite. This research is an extension of previous work done at the Air Force Research Laboratory located at Kirtland Air Force Base, New Mexico [1]. The design process, from initial concepts to test results using Application Specific Integrated Circuit (ASIC) standard library cells in a 4-bit microcontroller, is presented. The design was fabricated using the Taiwan Semiconductor Manufacturing Corporation 0.35-micron commercial process [2]. This ASIC called the Microdot, a 4-bit microcontroller, can be used for data storage, data transfer, and data manipulation.

1.2. Problem Statement

Power consumption and reliability are critical concerns for satellite design. A hierarchical decentralized control network architecture can potentially meet the necessary low power requirement and be tolerant of component failures. The Microdot 4-bit microcontroller is a small yet vital piece of the satellite network system's success. Many possible scenarios for Microdot implementation into a network system's architecture exist. However, one general scenario describes the Microdot's role in a system architecture: the network contains small microcontrollers, which run programs that can check for temperature or chemical variations thereby reducing power consumption. Normally, a larger microprocessor would be tasked to continually run programs to check multiple sensors, leading to high power consumption. An engineer can reduce the power consumption by designing the smaller microcontroller to check sensors and report back to a larger microprocessor only when an out of range sensor value has been detected. The program being run by the small microcontroller determines the out of range sensor value. The result of a small microcontroller, the Microdot, detecting an event and reporting back to the larger microprocessor is less power consumption for the satellite and less run time of the larger microprocessor. The power savings comes from only running the larger microprocessors when necessary for data transmission, microcontroller programming, and data storage.

In order to make the Microdot a reality, a design methodology must be followed. Many software programs help an engineer follow along the design path; however, even without software programs, an engineer must follow a distinct design regiment that contains hierarchical elements. These elements are the building blocks that make it possible to create a larger component, for example, the Microdot. If one is going to learn what the Microdot is, one must first learn under what design methodology the Microdot was created.

1.3. Methodology

A design methodology or design flow is a sequence of steps that are followed to make an ASIC. The steps are either classified into the logical or physical design. Logical design is design that involves describing functionality of circuitry. Physical design is the actual implementation of the functionality desired. However, some steps such as system partitioning may be considered as either logical or physical design [3]. This generic outline of the ASIC design methodology will be explained in detail in Chapter Two and Three.

The initial step in ASIC design is to define the function or behavior of the circuit. At the behavioral level, the operation of the system is captured without having to specify the implementation. Next, the design constraints such as area, power, and speed are established along with the type of Complementary Metal Oxide Semiconductor (CMOS) technology that will be used. The gate size selected by the engineer is a significant decision, because the gate size will have a direct impact on whether or not the design constraints can be met. The next step is crucial, which is deciding the architectural design. The architecture is broken up into small manageable blocks, in which all the interfaces and design constraints are identified. Each block is developed in the behavioral domain and tested to make sure functionality is correct. Within each of these blocks, the elements which make up the blocks are the lowest level of hierarchy. Interconnections between the elements within the blocks are defined, as well as inter-block connections between elements from different blocks. Each of these elements are tested for functionality because the lowest level of hierarchy must be correct in order for any of the blocks to work correctly.

After successful testing, the behavioral level of the design gets translated into the structural level, where the implementation of the design is decided. Similar to the behavioral stage, verification testing is performed at the structural level. Each element, block, and fully connected top-level product must successfully pass the simulated tests. Once successful, the design can be transformed to a physical layout. Again, verification must be accomplished for each block and element at the physical level before all the blocks can be pieced together to form the final top-level product.

The fully assembled circuit is simulated not only for functionality, but also for power and speed requirements. If the circuit fails to meet the design requirements or constraints, then the process is repeated until the circuit meets or exceeds the requirements. An engineer must decide before physical layout, which fabrication process will make the design a reality. One of the biggest considerations is the gate size or technology size to which the transistors will be created. The gate size will strongly influence power, speed, and area results in the final product. Once the design has been fabricated, the circuit can be tested and compared to the simulation results [4].

1.4. Overview

Six chapters comprise this thesis. Chapter One provides an overview of the thesis while introducing the problem and generic design methodology.

Chapter Two reviews the synchronous design flow and the history of 4-bit microcontrollers. It continues by comparing Field Programmable Gate Array (FPGA) with ASIC standard cell design and then repeats the differences between decentralized and centralized architectures and explains radiation hardening of electronics. The chapter concludes with a presentation of other design work related to this research area. Overall, Chapter Two highlights the problem's importance thereby providing motivation for this thesis.

Chapter Three applies the theories discussed in Chapter Two in order to develop a working design. The overall functionality and purpose of the Microdot are explained. The instruction set of the Microdot is explained along with data flow throughout the Microdot. An overview of the Microdot design including the architecture of the top-level design and its major elements are the primary focus of Chapter Three. Interfaces between the elements within each block are diagrammed. The hierarchical structure is broken down and explained in great detail. This chapter gives the reader a higher level construction of the Microdot.

Chapter Four explores the higher level design overview presented in Chapter Three at the elemental level. Chapter Four is also a presentation of the design implementation of each functional block. Each block and its respective elements introduced in Chapter Three are now revisited in detail. Pin labels of each element within the different blocks are displayed. How the different blocks interact with each other is described in detail. The final design of each block, as well as other possible designs, are presented.

Chapter Five reveals and analyzes the simulation and fabricated chip test results. Results are given at each level of design for the individual components as well as the results of the top-level design. The testing process and errors encountered along the way are explained. Each level of testing is described, which includes behavioral Very High Speed Integrated Circuit Hardware Description Language (VHDL) simulation, structural

VHDL simulation, IRSIM, and High Accuracy Simulation Program with Integrated Circuit Emphasis (HSPICE).

Chapter Six concludes by summarizing the collected results of the research. In addition, lessons learned during the design process and recommendations for future work in this area of research are discussed.

2. *Literature Review*

2.1 Introduction

This chapter presents background research needed to understand the history of 4-bit microcontrollers, the differences of between FPGA design and ASIC design, the difference between decentralized versus centralized architecture, and radiation hardening of electronics. The literature search did not find a 4-bit microcontroller built for a similar purpose as the Microdot. However, a variety of microprocessors and microcontrollers are summarized in a table to outline the functions and performance of both old and state-of-the-art designs.

2.2 Synchronous Design Flow

The design of synchronous microcontrollers is a very well defined process. The process flows through three domains: behavioral, structural, and physical, which are illustrated in Figure 2-1.

The initial step is to begin with a behavioral representation of the component. A behavioral representation describes how a particular design should respond to a given set of inputs [4]. The process starts by determining the behavior of the highest level of hierarchy and then describing the behavior. The description may be specified by Boolean equations, tables of input and output values, or algorithms written in standard high level computer languages or special Hardware Description Languages (HDLs). The latter include VHDL, Verilog, and ELLA [4]. There are many levels of abstraction within the behavioral domain. As an engineer descends through the levels, more detailed

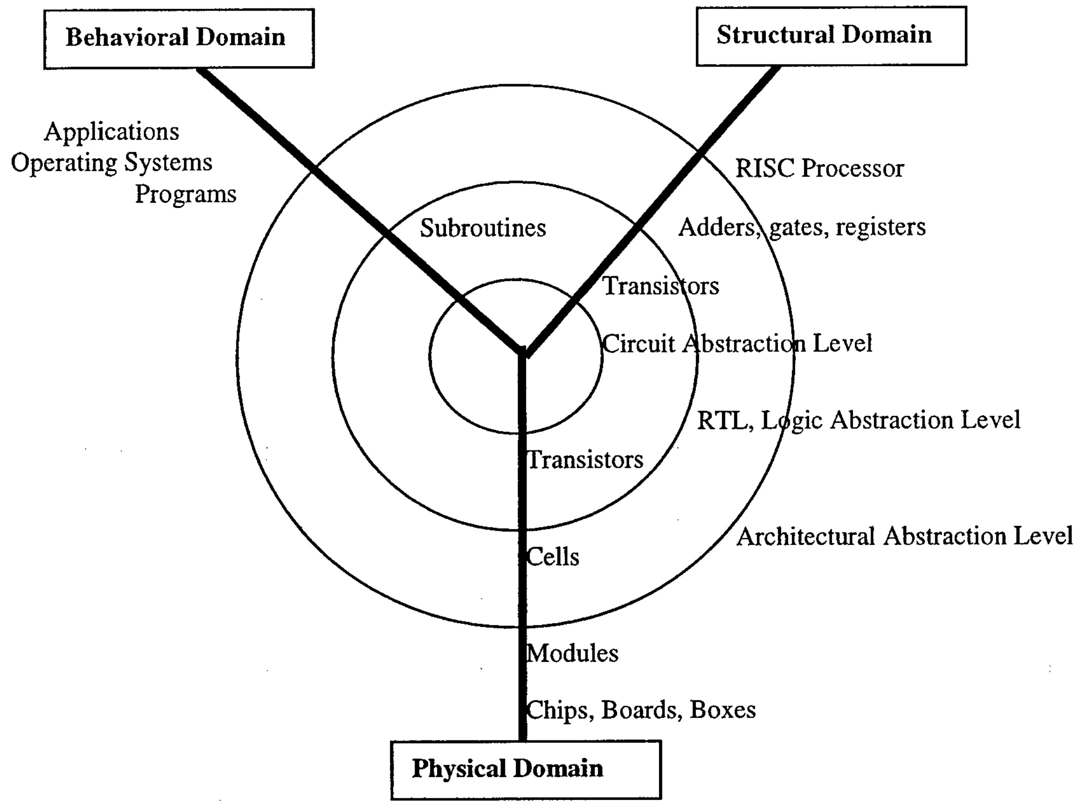


Figure 2-1. Diagram of Design Domains for ASIC Design from [4]

information about a particular implementation becomes evident. An example is one might start with an algorithm describing a system and further explain to a description of the specific hardware registers and the communication between them that is necessary to implement the algorithm. At the lower levels of abstraction, the Boolean equations to implement the algorithm would be specified. The goal of most modern design systems is to convert a high level specification into a system design in minimum time and with maximum likelihood that the system will perform as desired [4]. Once the behavioral domain is represented and described, an engineer is ready to cross over to the structural domain. This transformation from behavioral to structural can be done either using automated software or manually.

The structural domain involves creating a structural specification, which specifies how components are interconnected to perform a certain function. Basically, the structural description is a list of modules and their interconnections [4]. The structural domain has levels of abstraction, which include the module level, gate level, switch level, and the circuit level. In each successive level, more detail is revealed about the implementation. Once again, just like in the behavioral domain, an engineer starts at the highest level of abstraction and gradually moves inwards to the lower levels. In Figure 2-1, an engineer would start at the processor level and begin to break the processor down into manageable parts. Eventually, the engineer would reach the transistor level, which would be the lowest level of abstraction. Once this level is reached in the structural domain, an engineer can translate the knowledge gained in the structural domain to the physical domain.

In the physical domain, a physical representation for a circuit is used to define how a particular part has to be constructed to yield a specific structure and hence behavior. Similar to the behavioral and structural domains, various levels of abstraction may be defined for the physical representation. For example, the physical layout of a 4-bit adder may be defined by a rectangle or polygon that specifies the outer boundary of all the geometry for the adder, a set of calls to submodules, and a collection of ports. Each port corresponds to an input or output connection in the structural description of the adder. The position, layer, name, and width are specified for each port. The calls to the submodules are another hierarchical level that includes physical layouts of basic gate structures, such as an AND gate. An example of how Figure 2-1 was used for this thesis is described below.

The initial synchronous design process starts by determining the function of the Microdot and then writing behavioral VHDL code to describe the functions. VHDL is the Department of Defense (DoD) standard and was used throughout this thesis [5]. It is only necessary to be concerned with the behavior or functions that the circuit will have. Once the behavioral VHDL code was successfully tested, the cross over to the structural domain began.

Some of the structural VHDL code was created by using a VLSI tool called Design Analyzer made by Synopsys [6]. The tool takes the behavioral VHDL code and translates it into the structural version of the VHDL code; in this structural form, the type of each logic gate and interconnection is specified. Certain blocks or modules of behavioral VHDL code were not transferred over to the structural domain. However, one can use behavioral VHDL code mixed in with structural VHDL code to create the top-level interconnected component. Once this structural level is error free during testing, the design is ready to transfer from the structural domain to the physical domain.

The automated structural VHDL code can be entered into a different VLSI tool, Lager Octtools, in order to receive a physical layout in the form of the VLSI layout tool called MAGIC [7]. The particular blocks that were not automated over to structural VHDL code had to be manually laid out and connected to the other blocks in MAGIC. Once the entire circuit layout is placed into a pad frame, all the inputs and outputs are connected by a place and route tool and then tested. This physical layout is the final design and it is sent to the foundry for fabrication. The synchronous design flow is shown in Figure 2-2.

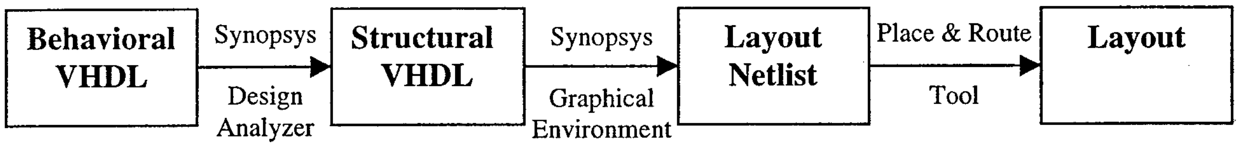


Figure 2-2. Synchronous Design Flow from [8]

2.3 History of 4-Bit Microcontrollers and Microprocessors

In 1972 Texas Instruments (TI) introduced the first 4-bit microcontroller called the TMS1000 [9]. The TMS1000 integrated a simple 4-bit microprocessor, a 1 kilobyte read-only-memory (ROM), and 32-byte random access memory (RAM) on a single chip. In 1974, the new microprocessor of the time to be used in electronic calculators was the Intel 4004. The chip measured 3.0 by 4.0 millimeters and used two thousand and three hundred transistors [9]. If the Intel 4004 were built using 0.35-micron process, it would be tenths of a square millimeter in area (without wire bond pads) and cost less than one cent to fabricate [10]. The microprocessor industry started with 4-bit microcontrollers and microprocessors in the early 1970's and continued on with 8-bit, 16-bit, 32-bit, 64-bit chips, and 128-bit chips [10]. As the microcontroller and microprocessor were developed, certain advantages and disadvantages became evident to design engineers.

The microprocessor and microcontroller are similar entities in the fact that they are made up of common elements. Some of those elements are SRAM, Arithmetic Logic Unit (ALU), registers, input/output ports, and control units. Typically, 4-bit microcontrollers are used in embedded applications, which are integrated with the memory and Input/Output (I/O) functions. These true single chip computers are low cost and can typically do all the work required in many simple control applications. Simple 4-bit and 8-bit microcontrollers control microwave ovens, computer keyboards, taxi meters, traffic lights, gas pumps, elevator control, medical instruments, vending machines, and digital scales for example, while sophisticated microcontrollers drive cellular phones and laser printers [9].

The microcontroller has the advantage of being a lower cost product than the microprocessor. However, the microcontroller does not have the flexibility of the microprocessor. The microcontroller normally takes less power to run than the microprocessor, because it is less complex. An engineer, depending upon the application, will first look to a microcontroller as a solution because of its less complex design. Some simple tasks that are listed above, such as traffic light operation can easily be handled by a microcontroller. The microcontroller can be reprogrammed also, just in case some type of variation is required throughout the device's lifetime. Reprogramming the traffic signal timing is a good example of how minor alterations may need to be made throughout the microcontroller's lifetime. In contrast, a microprocessor would be used in a more changing environment than an embedded application. Desktop computers, which take on many different applications and functions, contain microprocessors which are adaptable to many scenarios. Microprocessors typically have bigger memory and control elements, because of the wide range of tasks that are expected of them. Bigger elements yields a larger microchip area, which always results in a more expensive microchip.

The Microdot is designed to directly interface to a sensor or actuator in the space environment. A nybble is a 4-bit set of data. The on-chip program memory will have a capacity of one thousand and twenty-four nybbles in the form of internal Static Random Access Memory (SRAM). Memory will be expandable to four thousand and ninety-six nybbles with the use of an off-chip SRAM device. The data path is four-bits wide. The instructions are stored in bit-serial fashion. Microdot uses a stack architecture, which helps result in a compact implementation of the microcontroller. The modified stack is able to hold one hundred and twenty-eight nybbles and have the ability to reach a range of stack positions from the top-off-the-stack (TOS) down to the sixteenth position [11]. The stack pointer, which points to the current stack address that is being accessed, has the ability of being adjusted. A 4-bit set of data is added to the current stack address. Therefore, this gives the ability to reach into the stack memory sixteen positions or addresses down into the stack. If certain instructions in the Microdot are used in a correct sequence, all of the stack's one hundred and twenty eight addresses are able to be accessed. A further explanation of this instruction sequence will continue in Chapter Three.

2.4 FPGA Design versus ASIC Standard Cell Design

The initial design for the Microdot was done using Altera VHDL software. This software is an initial design step in behavioral domain towards creating an IC design using an FPGA [1]. Both FPGA and ASIC standard cell design have distinct advantages and disadvantages.

The advantages of an FPGA design are lower non-recurring engineering (NRE) costs and greater flexibility [12]. FPGA chips are sold from manufacturers and the customers program the chips to do their desired function. The compiler software usually only needs to know what function the FPGA needs to perform in order for the chip to be programmed [12]. This process takes much less time than going through a synchronous design process, where the designer has to step through each level of design [12]. Overall, less engineering design hours yields a lower NRE cost compared to an ASIC standard cell design [12].

FPGA chips have great flexibility, because of the easy access to correct design or logic mistakes. The customer can reprogram some types of FPGAs, after finding an error, as many times necessary to get the FPGA to function correctly. After an ASIC standard library cell design has been fabricated at a foundry, any errors found will most likely not be able to be fixed by the customer until a redesign and a new fabrication run is complete. FPGA designs clearly have a shorter turnaround time than an ASIC standard cell design [12].

ASIC standard cell design advantages include lower power consumption, faster processing speed, and greater area efficiency [12]. Many FPGA designs only use sixty to eighty percent of a given die, which results in unused area on the silicon wafer [12]. The ability to put an extra twenty to forty percent of functionality on the same size die helps to partition the design in order to minimize the connections to the outside world. The minimization of Input/Output (I/O) buffers can drastically reduce power consumption and increase speed by eliminating the capacitance associated with the removed I/O buffers [12]. Figure 2-3 summarizes the performance versus ease of modification tradeoffs made for different technologies including the FPGA and ASIC standard cell designs.

A full custom IC design involves an engineer manually laying out each individual transistor and element. The full custom IC design has the advantages of fastest speed, least amount of area, minimum power consumption, and least volume cost. However, the full custom design has the disadvantages of high non-recurring engineering costs and low flexibility. The standard cell IC design is a design involving a standard set of gates

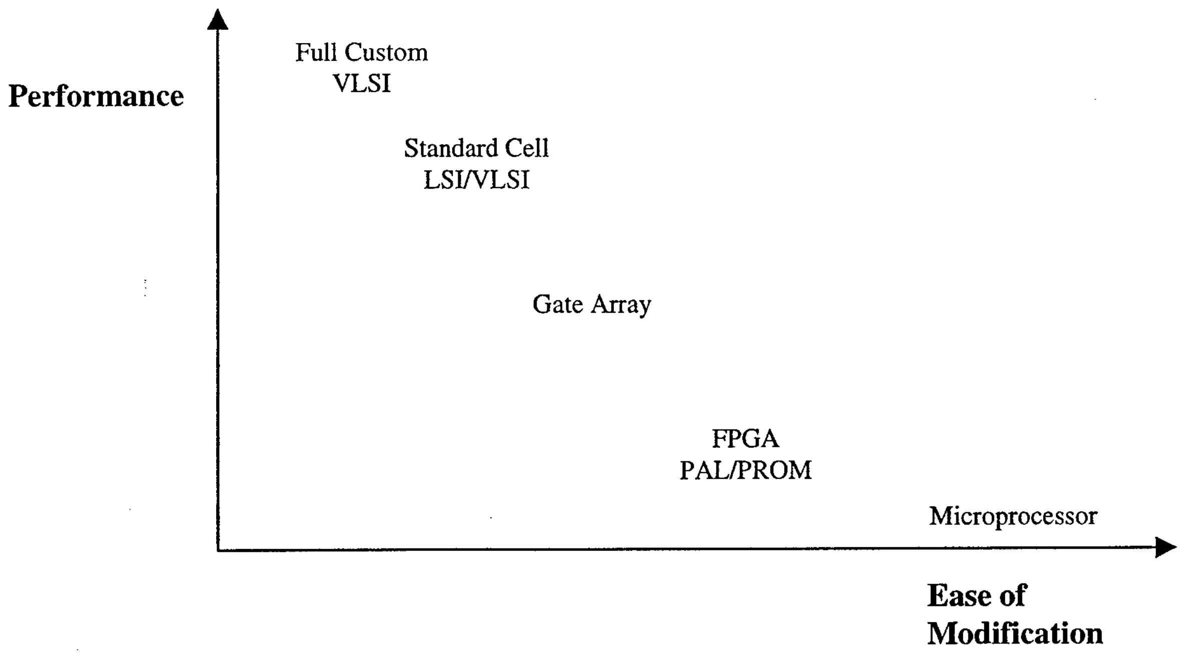


Figure 2-3. Technology Choice for Microprocessor Design from [13]

which have similar height dimensions, so they can be connected together easily. An engineer would build the standard cells or receive them from a foundry, but no modification would take place on the transistor level. An automated layout tool or manual connection of the standard cells would be done to create the integrated circuit. A standard cell design would be slower, take up more area, consume more power, and have a higher volume cost than the full custom design. However, the non-recurring engineering costs would be less and there would be more flexibility than the full custom design. Gate arrays are regular structures of repeating types of gates. There are three types of gate array structures including channeled, channelless, and structured. The channeled gate array is manufactured by only customizing the interconnections between all the gates in the array. There are channels set aside between each row of gates that are used for interconnections. A channeled structure has the advantage of being easy to design, layout, and route. However, the channeled gate array uses twice the area of the channelless gate array. The channelless gate array contains no set aside area for routing, but connects the gates using contact layers of metal that are laid on top of the gates. The gates that are not used are simply not contacted by the metal layers. The logic density is much greater than the channeled gate array, since area is not set aside for interconnections. Therefore, the advantages of the channelless gate array are smaller area and faster speed than the channeled gate array [3]. The disadvantage of the channeled gate array is the routing and interconnections are more complex than the channeled gate array. Finally, the structured gate array is an embedded gate array which sets aside some IC area and dedicates it to a specific function. The embedded gate array either can contain a different base cell that is more suitable for building memory cells, or it contain a complete circuit block, such as a microcontroller. The structured gate array has the advantage of a quick-turnaround time, improved area efficiency, lower cost, and increased speed and power performance. The disadvantage of the structured gate array is the embedded function is fixed, so flexibility is greatly limited. The gate array design falls in the middle of the rating for the performance and ease of modification factors. Programmable Array Logic (PAL), Programmable Read-Only Memory (PROM), and Field Programmable Logic Array (FPGA) fall closely together in Figure 2-3. The PAL and the PROM are classified in the family of Programmable Logic Devices (PLDs). PLDs are available in standard configurations that may be programmed for a specific application. A PROM is a device with a matrix of logic macrocells to be used as memory cells. A PAL is a device consisting of a programmable AND plane and a fixed OR plane [3]. By using combinations of different types of gates, the device can create numerous kinds of functions that the engineer requires. Some devices are programmed by applying high voltages or blowing metal fuses. The most useful devices for a changing environment is a reprogrammable PLD, which uses a high voltage to initially program and to erase old programs. These devices have a quick-turnaround, since they can be programmed rather quickly. However, there is a high volume cost for these devices and a lesser performance results from non-optimization of logic. An FPGA is a step above a PLD in complexity. There is very little difference between an FPGA and a PLD. Some types of FPGAs can be reprogrammed in the field, which yields excellent flexibility. The microprocessor is the easiest design to modify, but its performance is the worst. An extreme amount of flexibility is built into a microprocessor, because of the wide range of duties that it may need to perform. However, the drawback for making a component so diverse is that it is not exceptional in any one task. The obvious advantage of the microprocessor is flexibility. The disadvantages of a microprocessor are high power consumption, high non-recurring engineering cost, and large area. Depending upon the application, this may be the best type of design to go with, because it may not need to be replaced only just reprogrammed [3].

After deciding on which kind of design is best for the application, an engineer must determine what kind of architecture the final product is going to operate in. The system architecture can help lead to design solutions along the design process. An important feature of designing a microcontroller or microprocessor is what kind of environment or architecture the device or component will operate in.

2.5 Decentralized versus Centralized Architecture

A centralized architecture or system consists of a single processor with its own memory, peripherals, and even possibly a few terminals. In the early 1970's, centralized architectures or systems were prominently known as mainframes [14]. During that time, the economic climate maintained high prices for any kind of computer especially those using centralized architecture. A classical law made by Herb Grosch, computer guru, stated that the computing power of a central processing unit (CPU) is proportional to the square of its price [14]. However, due to today's microprocessor technology this law no longer holds true [14]. Nowadays, by paying twice the amount of money, a person will get basically the same CPU, but at a higher processing speed. Because CPUs can be manufactured inexpensively, it is much easier and cost effective to create a powerful system by connecting multiple CPUs in to a distributed or decentralized architecture.

It is easier to understand the premise for decentralized architecture by describing the internet. The internet uses network hierarchy to create a Wide Area Network (WAN), which is then connected to a Local Area Network (LAN), thus providing millions of people access to the internet at the same time [11]. System control is embedded into the lower hierarchical levels. This is described as embedded control, where each level in the hierarchy has enough control to report back up the chain of hierarchy if a problem arises. However, if no problems exist the lower hierarchical levels have enough power and control to manage their individual sectors.

A decentralized or distributed architecture has many advantages over centralized architecture. First, the price to performance ratio in a decentralized system is much better than in a centralized system [14]. A group of microprocessors is able to perform tasks that a single mainframe cannot [14]. A more powerful system can be built using a decentralized architecture (multiple CPUs) and usually at a lower cost. Another advantage of distributed systems is that many jobs are inherently distributed by nature.

One example is a vending machine company wanting to know the status of its vending machines all over a city. By placing small microcontrollers in the vending machines that will communicate a status back to a central location, the vending machine drivers can more accurately plan their route for the day and be more efficient [11]. Another example is taking in temperature data from different sensors all at different locations within a satellite. The satellite controller may just want to know overall temperature of the satellite, but also know local temperature data on a particular circuit board [11]. One of the most important advantages is higher system reliability. The workload is spread out across many microcontrollers in a distributed architecture so, if one breaks down, perhaps only 3% of the system performance is lost [14]. For a single mainframe, if a problem occurs the whole system could be down and unavailable. Errors still occur in a distributed system, but overall availability and reliability are higher because an error may not significantly affect the overall performance. The last advantage of decentralized architecture is the ability to expand the system when growth needs to occur. Since the system is already set up for multiple processors, adding another processor is less work, time, and money than restructuring a single mainframe or buying a whole new mainframe in a centralized architecture [14].

2.6 Radiation Hardening of Electronics

The term "radiation-hardening" originated from the military needing a type of electronics to operate in a radiation environment [8]. Many space and nuclear applications require some sort of radiation protection. However, due to high cost factors of radiation hardness, military electronics are designed to be "radiation-tolerant" versus "radiation-hardened" [8]. The difference between "radiation-hardened" and "radiation-tolerant" is dependent on the amount of radiation a part can withstand before failing or malfunctioning [8]. The specific amount of radiation protection defined to each term is different depending on the user (commercial or military engineer). The unit of radiation typically used is the "rad" [15]. A rad is equal to one-tenth a Joule per kilogram [15]. From a military perspective, 100 kilorad (Si) is considered "radiation-tolerant" and 1 megarad (Si) is considered "radiation-hardened" [15]. However, when discussing levels of radiation protection, one must keep in mind the various types of radiation. For example, by saying that a circuit is 100 krad (Si) tolerant to total ionizing dose radiation does not describe the tolerance for any other type of radiation [15]. Therefore, it might be necessary to perform radiation tests to determine the degree of dose rate or single event protection for that same part. There are many methods that can be used to protect a circuit from radiation and several of them are discussed in this section.

2.6.1 The Need for Radiation Protection

The U.S. Department of Defense and the United States Air Force need radiation-tolerant circuits. These circuits are used in the space and nuclear environment. The overall effort to make circuits radiation-tolerant is broken up into three categories of radiation exposure: long-term total ionizing dose, short-instantaneous dose rate, and single event effects [15]. I begin by discussing the effects of total ionizing dose on circuitry.

2.6.1.1 Total Ionizing Dose

Total ionizing dose is the accumulation of radiation, usually measured in rads, in a circuit over a long period of time. The radiation comes in the form of high-energy ions, gamma rays, X-rays, protons, low-energy protons, electrons, and neutrons. Radiation is induced by either a nuclear detonation, sun-induced solar winds, or a galactic-induced event in space. Electron-hole pair creation occurs in CMOS transistors when energetic particles bombard the CMOS circuitry. Silicon dioxide is used as an insulator underneath the gate junction and in between transistors [15]. The energetic particles disrupt the charge balance of the silicon dioxide (SiO_2) in the n-channel and p-channel transistors and create ionization paths. Typically, the SiO_2 in CMOS circuitry is broken up into two regions, the gate oxide and the field oxide [15]. The gate oxide is a thin high-quality oxide that separates the channel and the gate contact of the device. The field oxide is a thick low-quality oxide that separates the different levels of metal or polysilicon wire runs. At the SiO_2-Si interface in the gate region, there are dangling bonds formed from the lattice mismatch, which make electrically active interface states [15]. The electron-hole pair generation has a secondary effect of causing additional interface states at the SiO_2-Si interface [15]. Thus, an induced charge sheet is formed which affects the transistor's performance characteristics. Radiation-induced interface states affect the CMOS transistor in many different ways. These effects are the lowering of the transconductance, softening the drain current versus gate voltage curve, additional threshold voltage shifts causing turnaround, and generating 'slow states' which result in a slow drift of threshold voltage over time [15].

Electron mobility is higher than hole mobility; therefore the electrons sweep out of the oxide and leave behind trapped holes in the oxide [15]. The holes tend to migrate to the SiO_2-Si interface and create a positive image charge on the channel, which is equivalent to a positive voltage applied to the gate contact. This positive charge build-up reduces the threshold voltage for an NMOS transistor, thus moving the drain current versus gate voltage curve to the left as shown in Figure 2-4. An NMOS transistor will become easier to turn on, and with enough charge build-up a leakage current will occur in the channel, thus increasing the circuit's power consumption [15].

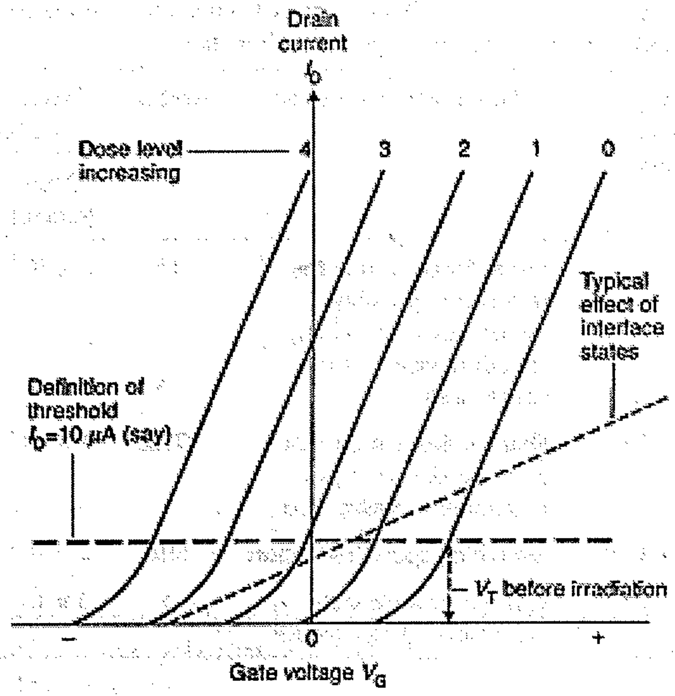


Figure 2-4. I-V NMOS Curve [15]

Another source of power consumption is channel formation in the bird's beak region of the NMOS transistor. The bird's beak region is where the gate oxide meets the field oxide at the edge of the NMOS transistor [16]. Positive charge build-up in the oxide forms a leakage channel between the source and drain region of the device, as shown in Figure 2-5. An additional leakage path can occur between neighboring NMOS transistors with charge build-up in the field-oxide. This leakage path will further increase power consumption for the CMOS circuit [15].

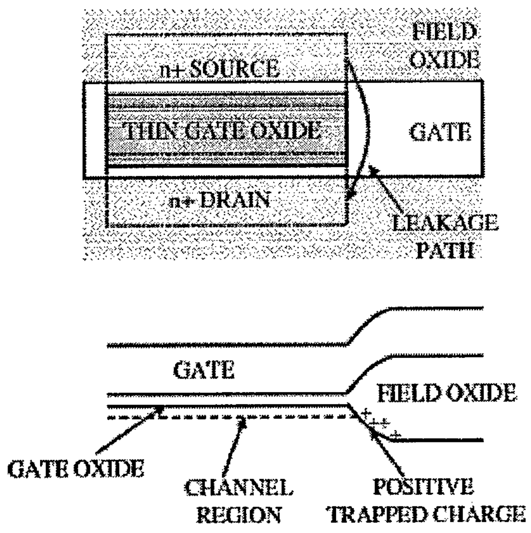


Figure 2-5. Bird's Beak Region in NMOS Transistor [17]

An opposite effect occurs in the PMOS transistor where positive charge build-up makes the transistor more difficult to turn on [15]. Therefore, the accumulation of charge at the SiO_2-Si interface shifts the drain current versus gate voltage curve to the left increasing the negativity of the PMOS transistor's threshold voltage, as shown in Figure 2-6.

With enough charge build-up, the threshold voltage can reach a value outside of the power supply's range and the PMOS transistor will not be able to turn on [15].

As total ionizing dose increases, the performance of the CMOS circuitry degrades. Also, as the total ionizing dose increases, the standby power supply current increases, which will eventually cause breakdown in a spacecraft due to the limited amount of available power from the power supply [15]. Figure 2-7 shows that as radiation increases the maximum operating frequency goes down, which results in slower operation of the microprocessor [18]. The decrease in operating frequency could cause failure in whatever system the microprocessor is responsible for [18].

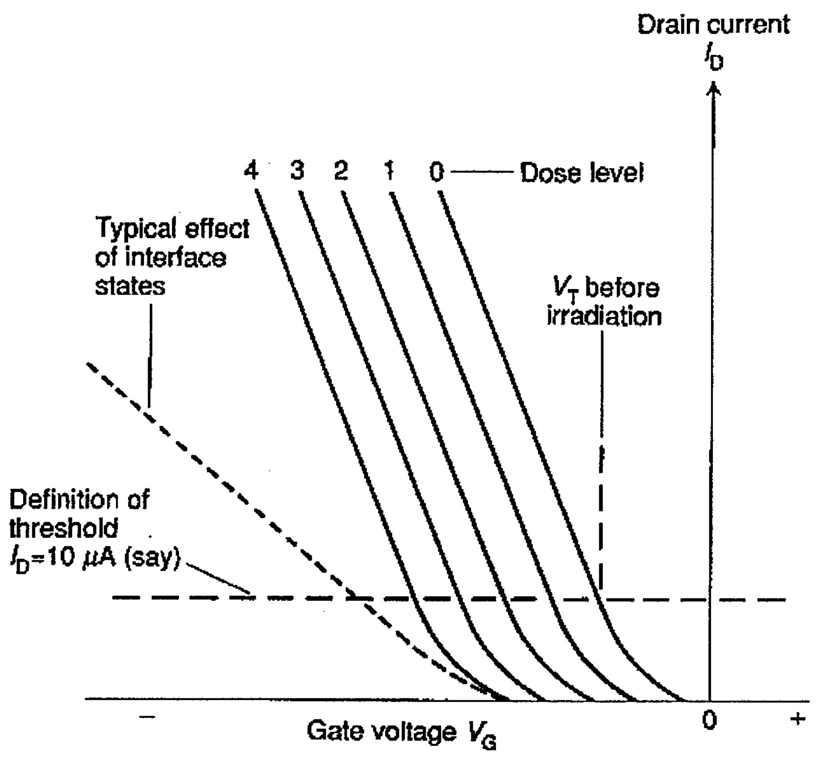


Figure 2-6. I-V Curve for PMOS Transistor [15]

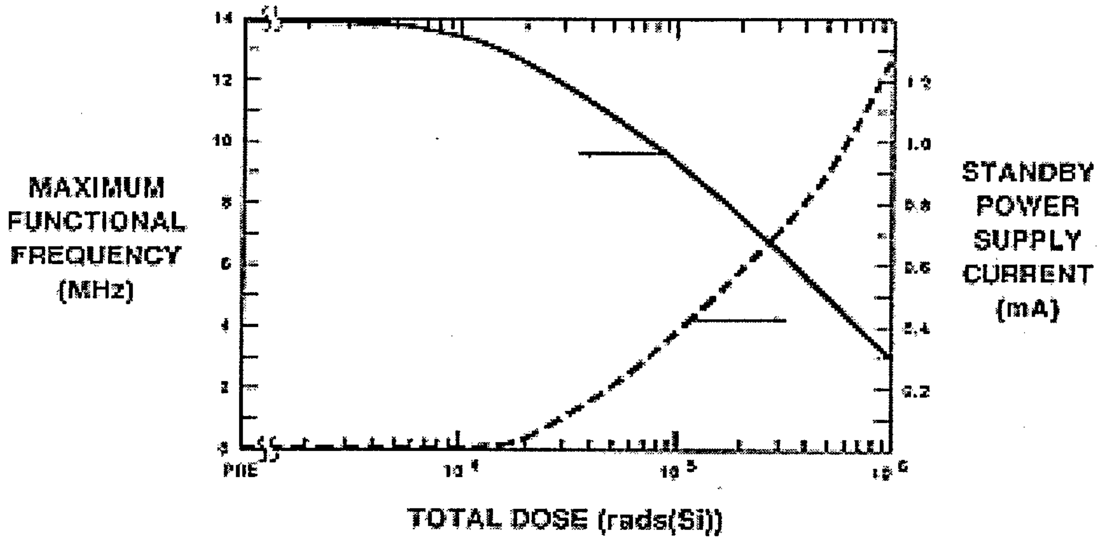


Figure 2-7. Microprocessor Trend for Operating Frequency and Standby Current versus Total Ionizing Dose [18]

2.6.1.2 Dose Rate

Dose rate is the amount of radiation taken in by the CMOS circuitry per second. Typically, dose rate effects are related to a nuclear detonation where a rapid time variation of particle radiation occurs in the CMOS circuitry. Besides nuclear weapon induced dose rates, there are lower dose rates caused by normal particle flow in outer space. Weapon dose rates will be in the range of 10^4 to 10^{12} rad (Si) per second and space background rates are on the order of 10 rad (Si) per second [19]. The main product of high dose rates from nuclear weapons is photocurrent generation throughout the circuitry. Extra current can generate rail collapses or voltage sags, which may cause device burnout [19]. Different dose rates have different effects on the CMOS circuits. Figure 2-8 illustrates, that at lower background dose rates found in the space environment, the threshold voltage shift is positive, where at higher dose rates the shift is negative [18].

The difference is the reduction of oxide-trapped charges at lower dose rates and an increase in interface states. Different oxide regions, depending on the value of the dose rate dominate the leakage current. At lower dose rates the gate oxide dominates the leakage current whereas at higher dose rates the field oxide is dominant as shown in Figure 2-9 [18].

2.6.1.3 Single Event Effects

Single Event Effects (SEE) are caused by a high energy particles that impact the CMOS circuitry. A single event is a one time occurrence that the circuit experiences. These particle strikes happen at particular points on the circuit board and are not a blanket strike encompassing the whole circuit board. There are four basic types of SEE,

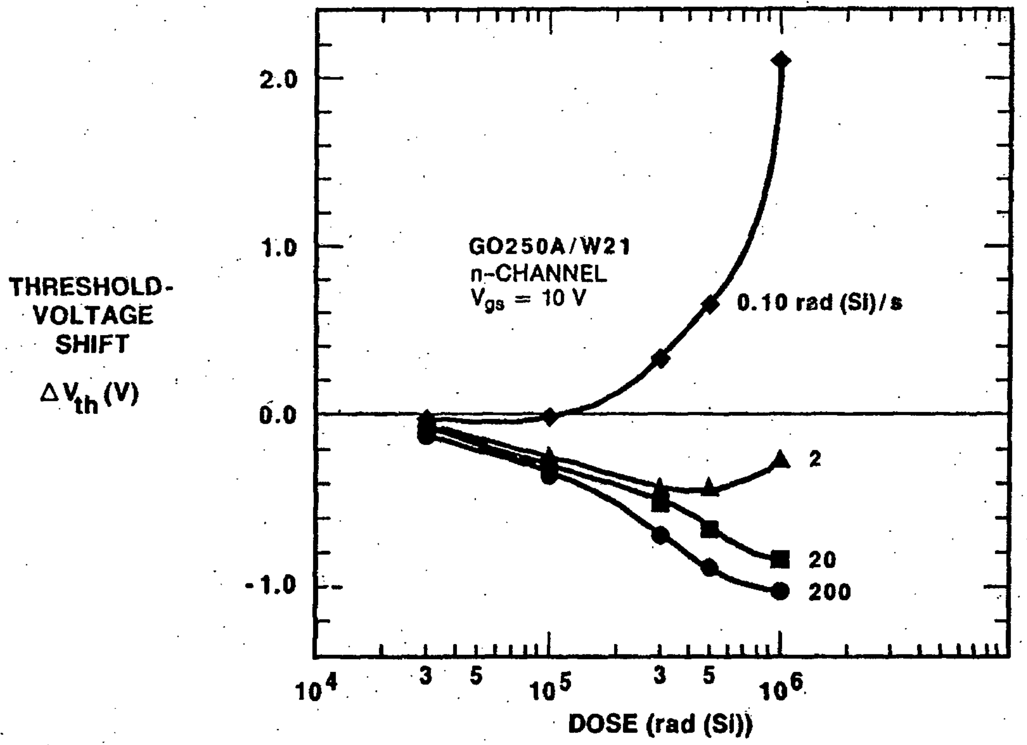


Figure 2-8. Threshold Voltage Shift in NMOS versus Dose Rate [18]

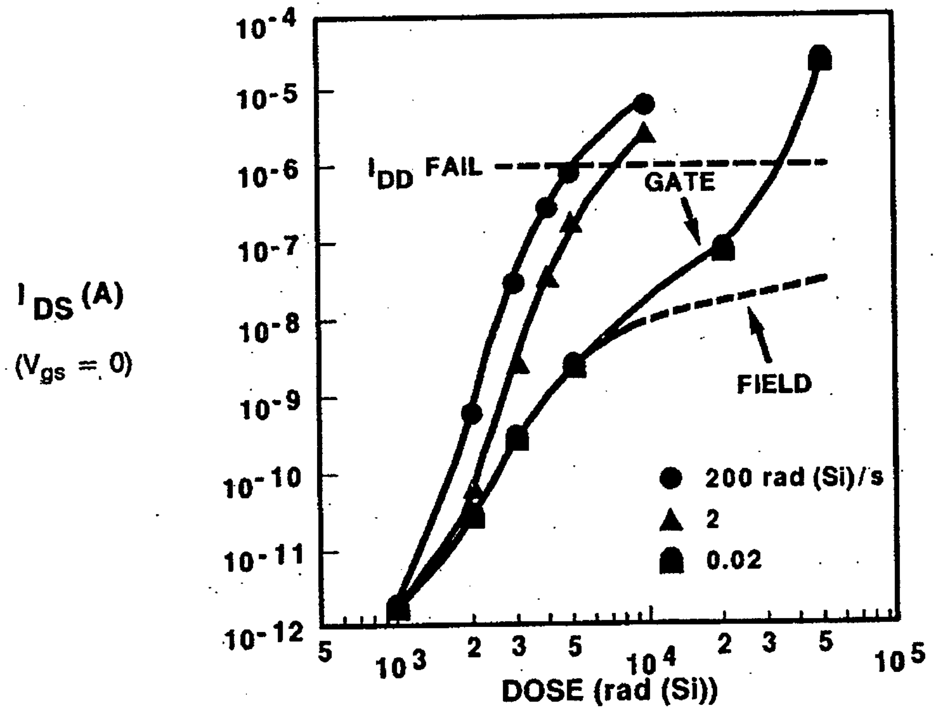


Figure 2-9. Leakage Current Dominance versus Dose Rate [18]

which are Single Event Upset (SEU), Single Event Latch-Up (SEL), Single Event Burnout (SEB), and Single Event Gate-Rupture (SEGR).

An SEU is caused by the energetic particle striking a sensitive node of the memory device as shown in Figure 2-10. SEU manifests by causing a bit-flip in memory devices.

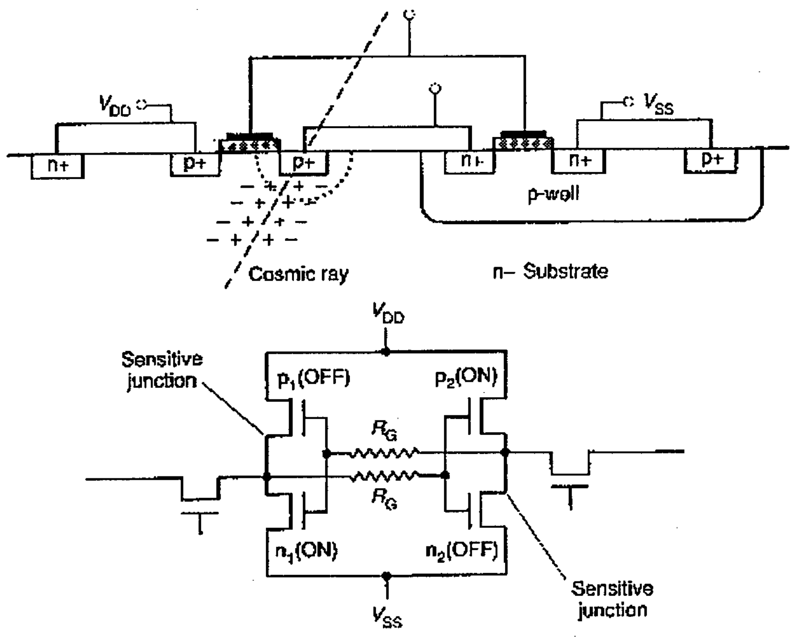


Figure 2-10. Sensitive Nodes for a SEU in a Typical SRAM [15]

The excess charge of the particle causes a memory cell to lose its current value and change the storage cell to hold the opposite value. This event is not permanent and the old memory value can be rewritten back into the memory cell [15].

SEL occurs when an energetic particle strikes the p-n junction containing the base region of the lateral parasitic p-n-p and n-p-n bipolar transistors located between an n-channel and p-channel transistor. The two parasitic transistors form a thyristor that self-reinforces itself to be in the "on" condition. This event is permanent and potentially destructive to the circuit. The only way to turn off the thyristor is by powering down the circuit and then reapplying power. The lateral parasitic bipolar transistors in the n-channel and p-channel transistors can be seen in Figure 2-11 [15].

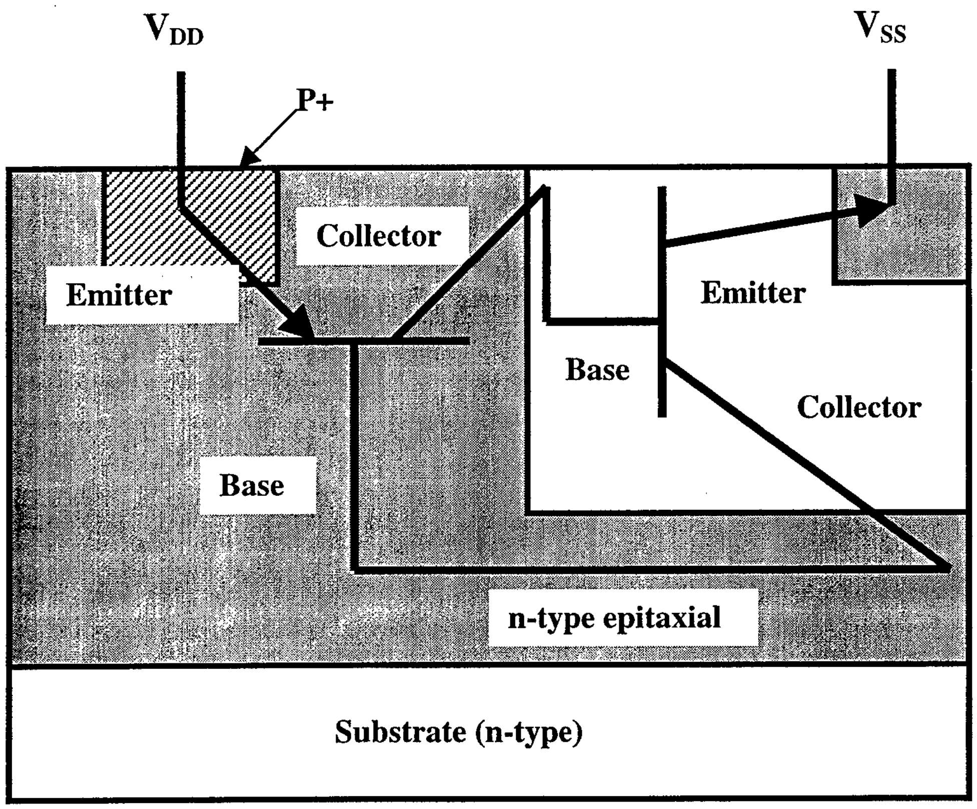


Figure 2-11. Lateral Parasitic Bipolar Transistors found in CMOS P-N-P-N Structure from [15]

SEB occurs when the drain-to-source voltage of the device exceeds the local breakdown voltage of the parasitic bipolar transistor; the device can burn out due to large currents and high local power dissipation. This event is of a permanent nature and cannot be repaired or worked around [15].

SEGR occurs when a heavy charged particle blows a hole through the thin gate oxide. Circuit failure is an eminent result of gate oxide failure. This event is also of a permanent nature and cannot be repaired [15].

2.6.2 Methods of Radiation Hardening

After reviewing the unfortunate effects of radiation damage, a clear motivation to protect against these occurrences arises. There are three different areas that can help CMOS circuitry be radiation-tolerant. First, shielding of a device will help stop some of the particles from reaching the CMOS circuitry or lower the particle's energy that does reach the circuitry. Next, fabrication of the wafers that the circuitry is manufactured on contains special characteristics to help protect against some radiation effects. Finally, layout techniques can smartly target areas of known weakness for CMOS circuitry and work to strengthen the circuitry overall [15].

2.6.2.1 Radiation Hardening through Shielding

Shielding aids CMOS circuitry by producing the following effects: stopping a particle; lowering a particle's energy; creating generation of secondary particles. Although, shielding is not the only solution to radiation hardening and cannot always be utilized. Many tradeoffs, such as thickness and increase launch costs, must be considered when assessing shielding. A typical thickness for shielding is 200-300 mils of aluminum (Al). Figure 2-12 illustrates that between 200-300 mils of Al is a reasonable thickness, because thicker shielding results in only a small increase in protection for an extremely higher cost. The result is less value for the dollar, therefore shielding only protects CMOS circuitry to a certain point [15].

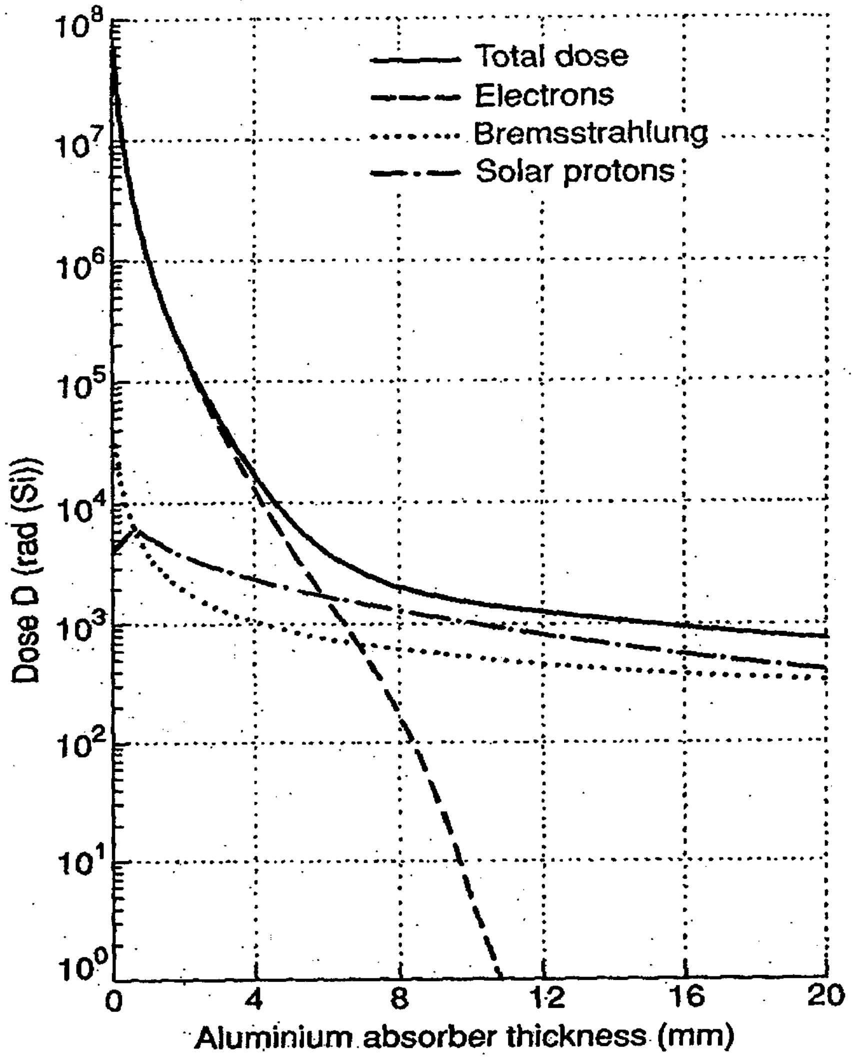


Figure 2-12. Shielding Advantage of Minimizing Total Ionizing Dose Radiation [15]

2.6.2.2 Radiation Hardening through Fabrication

Fabrication processes such as epitaxial-layer growth, Silicon-On-Sapphire (SOS), Silicon-On-Insulator (SOI), and quality oxide growth will help make CMOS circuitry radiation-tolerant.

To help suppress single event latch-up (SEL), the use of an epitaxial layer is a very useful fabrication technique. For the broader range of CMOS technologies, hardening against latch-up may be achieved by the use of a lightly doped epitaxial layer on a heavily doped (low resistivity) substrate. The low-resistivity substrate degrades the gain of the parasitic bipolar transistors and limits base-emitter junctions. Also, the substrate acts as an effective charge collector. Optimization of the resistivity and thickness of the epitaxial layer is important in order to achieve adequate immunity to latch-up. An example of what the epitaxial layer and normal processed bulk CMOS look like is shown in Figure 2-13 [17].

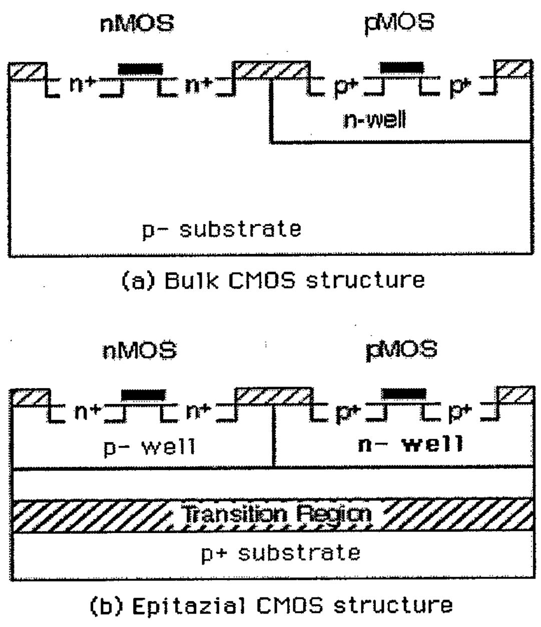


Figure 2-13. Bulk CMOS and Epitaxial CMOS Structures [17]

Another fabrication technique that helps with radiation hardening is SOS. SOS is a more complex form of dielectric isolation. A single-crystalline silicon film is grown over a sapphire substrate. The silicon island is doped to make an n-channel or p-channel transistor. Sapphire is a dielectric that has an inherently high tolerance to radiation. The sapphire protects the device against dose rate and single event effects. Leakage currents cannot flow between devices because the transistors are built on an insulating substrate. Therefore, guard rings that limit leakage current between transistors are unnecessary in SOS, and active devices can be packaged closer together. Also, there are no parasitic bipolar transistors to latch-up, because the n-channel and p-channel transistors are in complete isolation. Figure 2-14 illustrates the SOS CMOS structure [17].

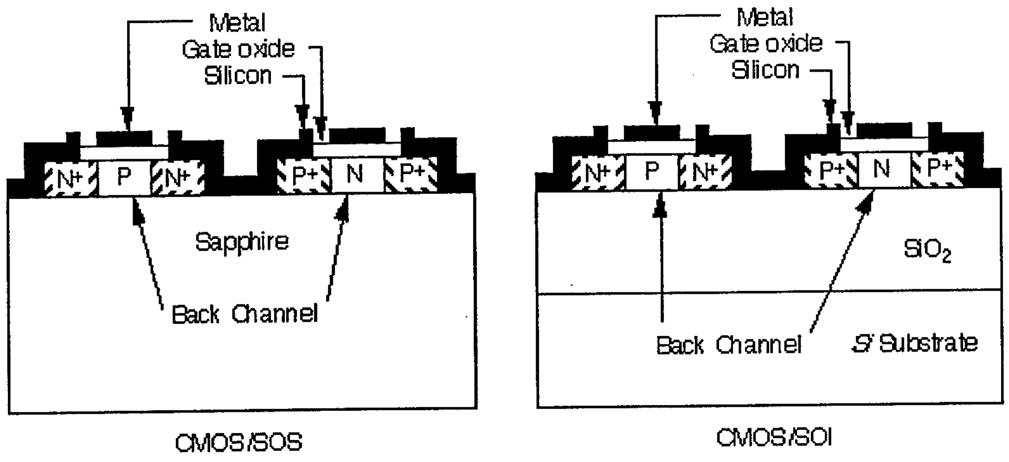


Figure 2-14. SOS and SOI CMOS Structures [17]

SOI technology is very similar to the process described above and used for SOS devices. Therefore, SOI and SOS have similar advantages. The only basic difference between them is the specific substrate used in each process. SOI uses SiO_2 for its substrate and SOS uses sapphire for its substrate. Silicon-on-Insulator devices can be fabricated using several techniques. One of those techniques is called SIMOX, or Separation by Implanted Oxygen [15]. In SIMOX, a high-current ion-implantation system is used to deposit a heavy concentration of oxygen molecules in a layer a couple of thousand angstroms below the wafer's surface. The wafer is then thermally annealed, and the oxygen forms a continuous SiO_2 layer beneath the silicon surface. The annealing process also is beneficial because it anneals damage in the top silicon layer caused by the implant. Therefore, a thin, high-quality layer of silicon is left on top of an insulating layer of SiO_2. This silicon is then ready to be used for device fabrication. Another helpful fabrication technique is building the transistors on a mesa. The process involves etching the silicon away between two active transistor areas and growing oxide in this etched region, thereby completely isolating the devices. The dielectric-isolation plane created by the SIMOX process enables increased circuit speeds and radiation tolerance. When utilizing this fabrication technique with smaller devices, it is important to keep in mind that a back-channel leakage can occur in n-channel devices as trapped positive charge build up occurs in the buried oxide [17]. Figure 2-14 illustrates the SOI CMOS structure.

2.6.2.3 Radiation Hardening through Layout

Several layout techniques assist in making a CMOS circuit radiation-tolerant. The way a transistor is drawn in the design process makes a difference in both the amount of leakage current and the single event susceptibility the overall circuit will have. Additional layout features such as guard rings, extra contacts, limiting fan-in and fan-out, and increased transistor sizing aid in circuit protection from radiation [17 and 20]. Two of the most common transistor layouts for radiation hardening are the annular and the dog-bone. Each of these special layouts is designed to protect against edge leakage in the bird's beak region.

The annular layout involves a drain region that is surrounded by polysilicon routing which forms a box shape around the drain region. The diffusion on the outside of the box shaped polysilicon is used as the source region. This layout has the advantage of eliminating the possibility for any edge leakage to occur in the bird's beak region of the transistor. With this layout, source-to-drain leakage can be avoided by forcing all source-to-drain current to run underneath the gate oxide, using an enclosed gate (or edge-less) geometry. Any current between the source and drain has to flow underneath the gate. Therefore, there is no current path underneath the field oxide or along the edge of the active area. The main disadvantage to this layout is the cost of increased layout area for the circuit [21]. Figure 2-15 illustrates the annular layout.

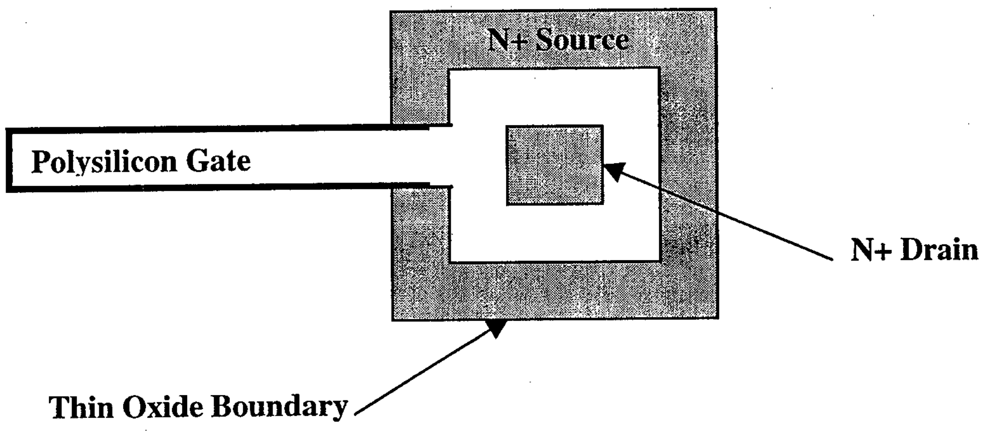


Figure 2-15. Annular Layout from [16]

The dog bone layout consists of a wider polysilicon edge the diffusion region. This creates a longer channel in the bird's beak region. By changing the effective width-to-length ratio, widening the polysilicon at the edges decreases the amount of leakage current. Although this reduces the leakage current it does not completely eliminate it. Disadvantages of this design include a decrease in the effective width of the intrinsic transistor, an increased gate capacitance, increased area, and difficulty and complication of the layout [21]. Figure 2-16 illustrates the dog bone layout.

Guard ring structures are heavily doped diffused regions that encircle the well and therefore prove to be another helpful radiation tolerant layout technique. They are very effective in preventing latch-up. Two types of guard rings exist, minority carrier guards and majority carrier guards [20]. Since CMOS devices form channels in the gate region

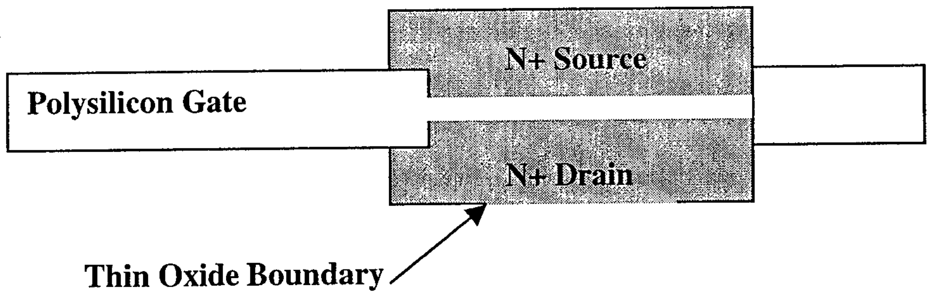


Figure 2-16. Dog Bone Layout [16]

using minority carriers in the well, the focus will be on the minority carrier guard rings. Minority carrier guards have an opposite doping type to that of the region in which they are formed. Therefore, they are able to collect injected minority carriers before they can cause a fault or upset in the circuit. Guard rings are placed in the substrate outside the well edge of a p- and/or n-well with frequent contacts to the rings. This reduces the parasitic resistances. Inherently, a guard ring eliminates the inter-transistor leakage current that occurs when the field oxide charges from the collection of excess charge and by increasing the spacing between neighboring transistors, thus lengthening the leakage path. As the length of a leakage path increases, it takes more excess charge or radiation to start or connect the two ends of the path, which causes leakage current. Figure 2-17 illustrates a CMOS inverter with a guard ring implementation [20].

Typically, commercial CMOS circuits designed for high packing density and high speed will minimize the space between n-channel and p-channel sources, and will use infrequent well and substrate contacts. The increase of well contacts will reduce latch-up susceptibility. The well contacts should be connected to the supply voltage (Vdd) or ground (Vss) to collect any injected charge. Plus, adding additional substrate contacts will help stabilize the transistors when they encounter a single particle strike. A lambda is a unit of length used in the VLSI CAD tool called MAGIC. The visual interface for

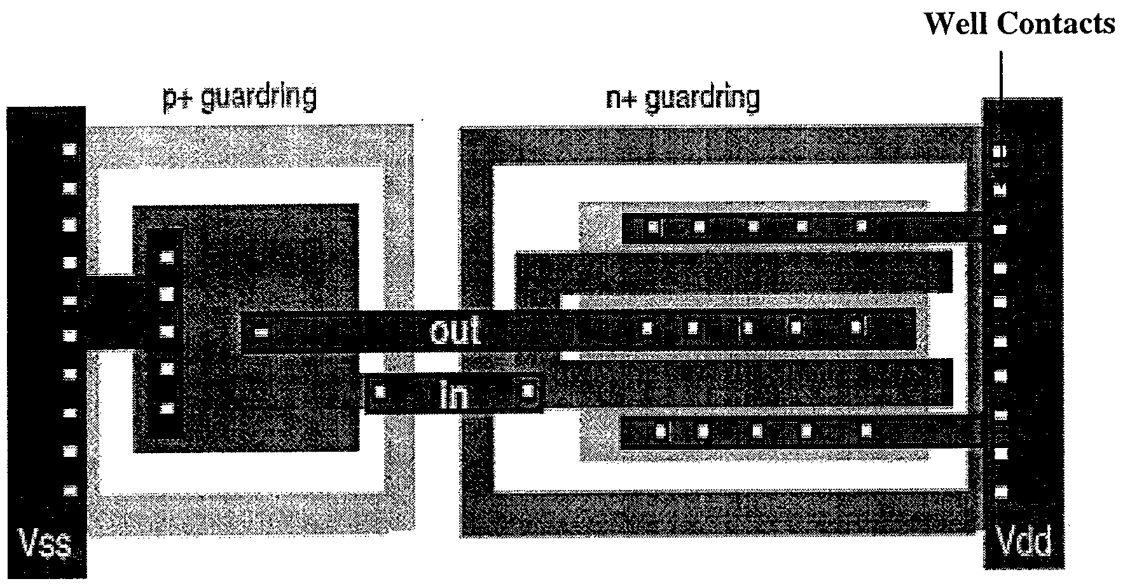


Figure 2-17. Guard Ring Structures [20]

MAGIC lays out a grid pattern broken up into squares, where each square equals one lambda. The contact spacing (body tie) should be no more than two squares or lambda apart in the circuit layout and should be placed between transistors to help deter latch-up. The extra body ties at the p-well/n-well interface will increase single event upset performance [20].

The terms fan-in and fan-out refer to the number of gates connected to the inputs and the output of a combinational logic gate. Typically, commercial CMOS circuits are designed to handle voltage fluctuations and to provide increased drive strength for the operation of running high fan-in and fan-out. The demand put upon high fan-in/fan-out transistors in order to drive certain large loads may not be realistic once exposed to some total ionizing dose radiation. Fan-in/fan-out limitations depend on the available technology, but it is a sound engineering principle to keep the fan-in/fan-out to a minimum [20].

Since it is known that the threshold voltages will change for n-channel and p-channel transistors following exposure to radiation, it is important to try to plan ahead for the expected post exposure changes. Electron mobility is higher than hole mobility, thus producing unequal drive strengths between NMOS and PMOS transistors. Therefore, some threshold voltage variance should be designed into the circuit and transistor PMOS to NMOS width ratios should not reflect the typical 2:1 ratio. A safety margin is designed-in to increase the PMOS width to NMOS width ratio to 3:1. However, with all the modifications designed to keep the transistor functional after radiation exposure the transistor still needs to be able to operate under normal conditions (i.e. at the beginning of the circuit's operational lifetime when total ionizing dose is zero). Figure 2-18 illustrates the width and length of a CMOS transistor. By changing the size of the polysilicon gate, the drive strength of the transistor will change. Typically, the length remains constant and, to either increase or decrease the drive strength, the width will be increased or decreased.

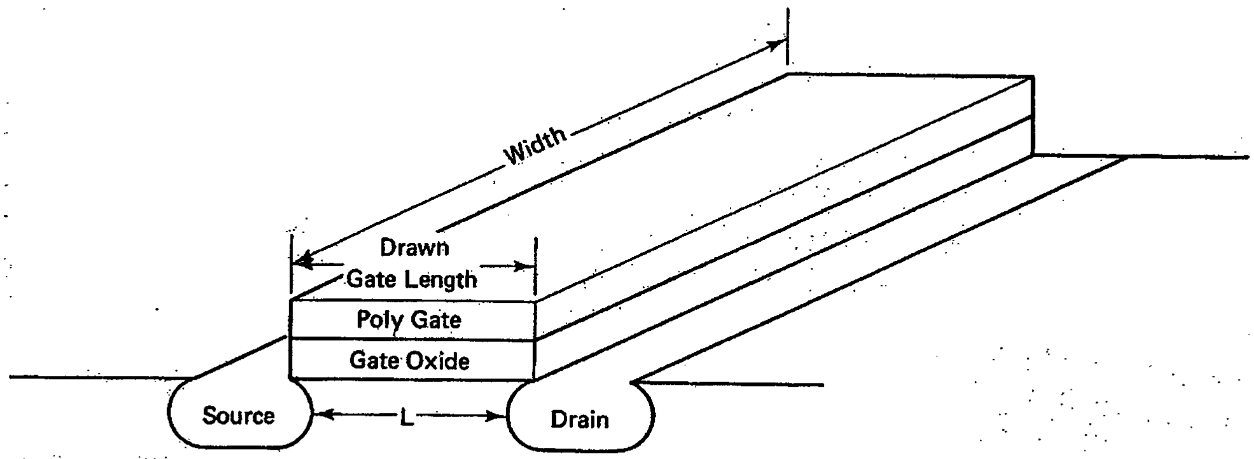


Figure 2-18. Width and Length of a CMOS Transistor [12]

2.6.3 *Previous Research*

Previous research on the Microdot was accomplished by Dr. Greg Donohoe and Mr. Jim Lyke of Air Force Research Laboratory at Kirtland Air Force Base, New Mexico. Dr. Donohoe and Mr. Lyke wrote the behavioral VHDL for the Microdot, which was broken up into five modules. The five modules are the following: control; program memory; arithmetic logic unit (ALU); modified stack; input/output module [1]. Their design implementation used FPGAs for the control module. An instruction set and modified stack architecture implementation was adapted from older stack machines such as the Forth and Java Virtual Machine [1]. This architecture definition and description are adequate for this implementation; however, in this effort, the five modules are implemented using an ASIC radiation-tolerant standard library cell design.

2.6.4 *Performance Comparison*

The key reason to compare design and performance parameters between the historical 4-bit, 8-bit, 16-bit devices is to set appropriate design goals for the ASIC Microdot. The Microdot did not have set design parameters for power delay product, speed, and size. However, due to the nature of the space application, the order of importance for the design parameters is power delay product, size, and speed. Speed is of minimal importance because of the nature of the application the Microdot will perform. By sampling temperature or chemical sensors, readouts do not have to be made at an accelerated processing speed such as 50 or 100 MHz. Simply to sample data once a second would be enough useful information for the ground station to have for a satellite. By slowing the clock speed down to the Hz range rather than kHz, the Microdot is able to operate while consuming less power. The minimization of the power delay product is the main goal of the Microdot. Table 2-1 shows 1993 4-bit microcontrollers performance parameters and displays current Microdot goals. These goals are based on previous microcontroller parameters and by adding in the advantage of the 0.35-micron TSMC process. A definite disadvantage over commercial microcontrollers for the Microdot will be making the microcontroller radiation-tolerant, which will increase area and power consumption. The area of the design should be optimized, in order to minimize power consumption and increase efficiency.

Table 2-1. 4-bit Microcontroller Design Parameters from [22]

Serial #	Manufacturer	Part Number	Power Consumption	Operating Frequency
1	Fujitsu	MB88xxxH	>100 microwatt range	666.7 kHz
2	OKI	MSM505x	4.5 microwatts	8 kHz
3	Sharp	SM530	18 microwatts	11 kHz
4	Sharp	SM500	60 microwatts	16.4 kHz
5	TSMC/AFIT	Microdot	< 50 microwatts	at the Hz or low kHz range

Another performance parameter, which specifically has to do with the space environment, is radiation tolerance or hardness. The mission needs and requirements of the United States Air Force will set the goal for the Microdot's radiation tolerance value. Table 2-2 illustrates some radiation information on different microprocessors and also displays the minimal radiation tolerance goals for the Microdot.

Section 2.6 involved discussions about the different types of radiation, radiation effects on circuitry, radiation protection for electronics, and finally goals for the ASIC Microdot design. Total ionizing dose, dose rate, and single event effects were explored in detail. Also, radiation protection through shielding, fabrication, and layout were investigated. Finally, radiation goals were discussed for the Microdot design.

Table 2-2. Ionizing Dose, Dose-Rate, and SEU response of ca, 1990 representative microprocessors from [19]

Ser.	Microprocessor	Manufacturer	Technology	Ionizing Dose Failure Threshold Krad (Si)	Ionizing Dose Failure Threshold @ Rads/s	Dose Rate Threshold (10^9 Rads/s)	Dose Rate Pulse Width (ns)	Threshold LET (MeV*cm^2/mg)
1	SA3000/80C85RH	Sandia/HA	CMOS-Epi	1000		1.9	20-50	120
2	GP 501	RCA	CMOS/SOS	1000		130		
3	SA3300/NS32016	Sandia/NSC	CMOS-Epi	5000	27.8	1	1000	30/120
4	80C186	INT	CMOS	8	125			
5	68020	MOT	CMOS	3	150			
6	GVSC	IBM	CMOS/SOS/SOI	3000	76.8	1/1E3	30	120
7	1750 A	RCA	CMOS/SOS	1000		1.0	20	8.5
8	GP001	RCA	CMOS/SOS	1000	112	45	20	75
9	80C85	HA	CMOS-Epi	100		0.3-0.5		75
10	TMP 320C25	TI	CMOS-Epi	52.6	207	0.26	20-50	
11	80C86	HA	CMOS-Epi	4	39.3	0.1	35	5
12	Microdot	TSMC/AFIT	CMOS-Epi	250		at least 10-3		

2.6.5 Conclusion

The goal of this thesis is to design a low power, small, and efficient 4-bit microcontroller able to operate in a space environment. This chapter covered synchronous design flow, the history of 4-bit microcontrollers and microprocessors, FPGA versus ASIC standard library cell design, decentralized versus centralized architecture, and radiation hardening of electronics. The last section compared performance parameters of several kinds of microprocessors and microcontrollers developed for commercial use and for the space environment. The key motivation for the Microdot is to improve on these previous performance and design parameters using the 0.35-micron process along with radiation hardening techniques, to improve spacecraft mission lifetime and safety.

3. Design Overview

3.1 Design Constraints

There are two key issues of design involving the Microdot microcontroller. First, an important task to be undertaken is optimizing the Microdot for minimum power consumption. Since the Microdot is programmable, inherently it can be used in many different scenarios. For example, a satellite has a limited amount of power that is produced by the solar panels of the satellite. As time goes on and the solar panels degrade due to radiation effects in the space environment, the total amount of power available to the satellite operations decreases. The Microdot microcontroller needed to be designed for minimum power consumption in order to achieve a particular satellite's mission lifetime. The final issue that played a part in the design of the Microdot was the actual size that the layout would yield. This issue connects to the previous issue of power consumption, since a larger microchip would need more power to operate due to larger capacitances throughout the circuitry. A goal was set for the Microdot to not be larger than 5 square millimeters or 2.2 millimeters on a side. The area goal was set to keep power consumption low and the design as compact as possible. The requirement for the Microdot to contain on-chip Static Random Access Memory (SRAM), which takes up a large amount of area, presented the biggest challenge. Chip area needed to be as small as possible without constricting routing between different elements in the Microdot. All of these design constraints were taken into consideration when designing the Microdot.

3.2 Microdot Design

The Microdot design is of a hierarchical nature. The Microdot microcontroller is made up of seven blocks and an expandable SRAM unit that is located off-chip. The seven blocks are made up of forty-five elements. The seven component blocks are broken into different functional areas that the Microdot needs and has to perform its overall mission.

The seven blocks are the Program Memory, Stack, Arithmetic Logic Unit (ALU), Input and Output (I/O), Data Acknowledge, and the Status Multiplexor. Each functional block was created to improve the design, test, and building of the Microdot.

The Program Memory is the block, which stores the programs to run the Microdot. It also keeps track of the memory address for the on-chip SRAM and the expandable off-chip SRAM units. In addition, it is responsible for sending out instructions and other types of information.

The Stack is the block, which is capable of storing data from different blocks and off-chip. It also keeps track of the stack memory address by the use of a stack pointer. The stack pointer points or directs data flow to a memory address within a stack.

The ALU block performs different kinds of arithmetic operations on data from the Stack block. In addition, the ALU block sends status signals to the Control block.

The I/O block is responsible for controlling bi-directional ports, which can be either used as an input or an output.

The Data Acknowledge block sends handshaking signals for programming purposes off-chip to whatever device is programming the Microdot.

The Status Multiplexor block is used to test the fabricated Microdot once back from the foundry. It is not necessary for the Microdot to run its tasks, but helps the designer see into the numerous data lines within the Microdot. The Status Multiplexor block adds to the design-for-test functionality, which is built into the Microdot. The hierarchical nature of this microchip design, under which less errors occurred and productivity increased, yielded a system of checks that occurred at different hierarchical levels. This hierarchical structure for the Microdot is shown in Figure 3-1.

The Microdot can be programmed from any device that can send a program and communicate with handshaking signals. Microdot performs 23 different instructions and does not incorporate any special provisions for interrupts or subroutines. It is designed to perform one task, which would be sensor data input manipulation, recording, and reporting of data to a master microprocessor within a hierarchical network. Microdot is capable of controlling an off-chip SRAM unit, which expands the 4-bit memory allocations from 1024 to 4096. The first 1024 memory addresses are located in on-chip SRAM and controlled primarily by on-chip resources. The Microdot is a self-contained microcontroller capable of running small programs that are less than or equal to 1024 4-bit instructions. However, it can accommodate larger programs up to 4096 instructions with the help of an off-chip SRAM unit. Microdot was intended to be connected to a master microprocessor, which would program the Microdot and read sensor data through four data lines.

It is crucial to understand the instructions the Microdot can perform, before understanding actually how the Microdot was built to perform these instructions. An operand for the Microdot contains 4-bits, since the Microdot is a 4-bit microcontroller.

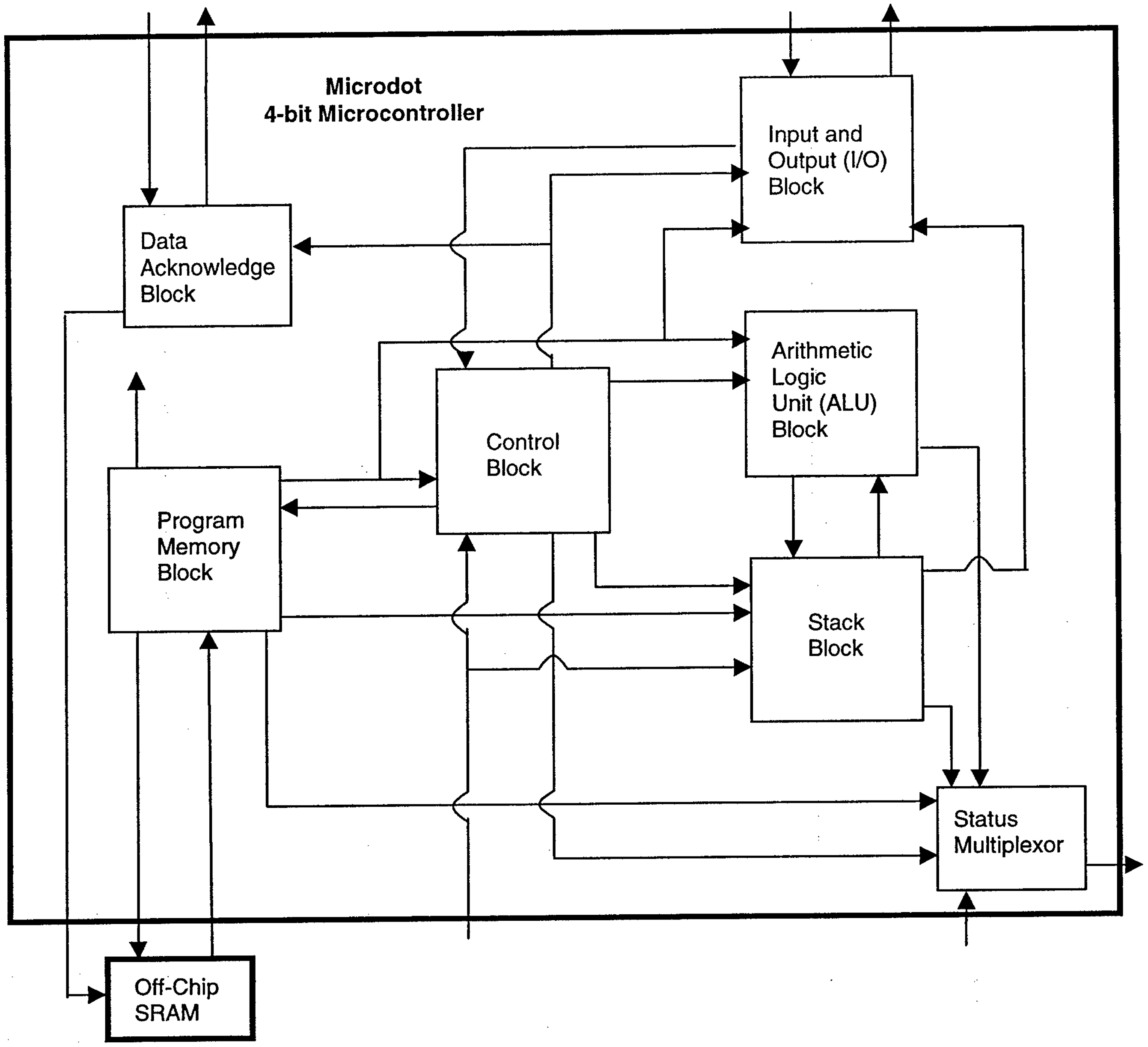


Figure 3-1. Microdot Hierarchical Structure

The Arithmetic Logic Unit (ALU) instructions contain two types of instructions. There are two operand instructions and one operand instructions. First, the two operand instructions are Add, Subtract, Add with Carry, Subtract with Carry, And, Or, and Exclusive-Or (XOR). The single operand instructions are Not, Shift Left, and Shift Right. The And, Or, and XOR instructions are bit-wise operations. In addition, the Shift

Left and Shift Right instructions use the Carry Bit as the data bit to shift into the operand and the data shifted out of the operand becomes the Carry Bit. The Stack instructions contain pop, push, load, store, duplicate, swap, and pick. The POP instruction simply reads the data at the top-of-the–stack (TOS) address onto the output data lines of the stack. The opposite of the POP instruction is the PUSH instruction, which writes data from the TOS register to the SRAM cells selected by the stack address. The LOAD instruction latches the data input lines, which are selected as inputs by the mask register into the TOS register. The STORE instruction sends the TOS register 4-bit output to the output lines of the Microdot, however each data line needs to be declared an output by the mask register in order to be sent off-chip. The mask register is an element, which simply sets the four bi-directional ports to either an output port or an input port. If a bi-directional port is set to be an input port and data gets sent to the port for output, the data will not be outputted and sent off-chip. The DUPLICATE instruction (DUP) simply duplicates the TOS element from the TOS register or TOS cache into the current stack address. The SWAP and PICK instruction can be used together to reach any 4-bit data set in any one of the 128 stack addresses. The SWAP instruction exchanges the TOS with any element within the stack (TOS-n). The PICK instruction copies the (TOS-n) data to the TOS. The next set of instructions has to do with the input/output module of the Microdot. The SETIO Mask instruction will set control bits for the input/output lines and force each line to be an input or output based on the particular control bit value for the line. The WAIT instruction sets the Microdot is a passive mode until an input line connected to some element off-chip produces a different value on the line than the TOS line has. Once a different value is detected on any of the four lines, the waiting is over for the Microdot and the next instruction is loaded. Finally, there are four miscellaneous instructions, which do unique tasks to specific Microdot elements. The LOADRAM instruction will write the data on the TOS register output lines into the stack corresponding to the stack address. The SKIP instruction is set up to increment the program memory address by as many as three times in order to skip up to three instructions. The JUMP instruction allows the program memory to jump to any of the 4096 addresses. The CLEAR STATUS REGISTER (CLRSR) instruction simply clears the status register bits to all zeros. Table 3-1 visually simplifies what the different instructions in the Microdot do.

3.2.1 Program Memory Block

The program memory block contains 10 of the 45 total elements for the Microdot. The hierarchical structure of the program memory block is shown in Figure 3-2. The program memory block has a number of functions that are crucial to the Microdot operation. This block is responsible for writing and reading 4-bit instructions from anyone of 4096 addresses along with getting the instructions out to the rest of the Microdot components. In addition, this block keeps track of the memory address, to be read from or written to, for the on-chip and off-chip Static Random Access Memory (SRAM) components. Either control lines from the Control Block or off-chip lines that come into the Microdot as inputs control all Program Memory elements.

3.2.1.1 Program Counter

The Program Counter is made up of several elements, which include a 12-bit adder (Program Logic), 12-bit register (Program State Machine), and a 12 line 3 to 1 multiplexor (Program MUX). The multiplexor takes in the incremented address, the jump

Table 3-1. Microdot Instructions

Instruction	# of Operands	Action	Instruction Type	Comments
Add n	2	TOS(value) <= TOS(value) + [TOS(value)-n] + Carry Bit	ALU	0≤n≤15
Subtract n	2	TOS(value) <= TOS(value) - [TOS(value)-n]	ALU	0≤n≤15
Add w/Carry n	2	TOS(value) <= TOS(value) + [TOS(value)-n] + Carry Bit	ALU	0≤n≤15
Subtract w/Carry n	2	TOS(value) <= TOS(value) - [TOS(value)-n] + Carry Bit	ALU	0≤n≤15
And n	2	TOS(value) <= TOS(value) and [TOS(value)-n]	ALU	0≤n≤15
Or n	2	TOS(value) <= TOS(value) or [TOS(value)-n]	ALU	0≤n≤15
Exclusive-Or n (XOR n)	2	TOS(value) <= TOS(value) xor [TOS(value)-n]	ALU	0≤n≤15
Not	1	TOS(value) <= not TOS(value)	ALU	
Shift Left (SHL)	1	TOS(value) <= left shift of TOS(value) thru Carry Bit	ALU	Carry Bit gets shifted in and gets replaced by the bit that is shifted out
Shift Right (SHR)	1	TOS(value) <= right shift of TOS(value) thru Carry Bit	ALU	Carry Bit gets shifted in and gets replaced by the bit that is shifted out
Load RAM	1	Stack TOS(value) <= TOS Register (value)	Stack	Loads Output of TOS Register into the Stack's Current Address
Pop	1	TOS(value) <= TOS(value)+1	Stack	
Duplicate (DUP)	1	TOS(value)+1 <= TOS(value)	Stack	
Load	1	TOS(value) <= Input(value) and IOMASK(value)	Stack	TOS Register gets the inputs if the IOMASK bits are set to '0'
Store	1	Output <= TOS(value)	Stack	Only bits where IOMASK is set as an output
Push c	1	Push constant onto stack	Stack	0≤c≤15
SetIO Mask (SETIO)	1	Write mask value 'm' to IOMASK	Input/Output	1=>output; 0=>input
Wait Mask (WAIT)	1	Wait for an event on I/O channel	Input/Output	Wait for change on sensitive input channels
Skip Mask (SKIP)	2	Skip if Status Register Bits and CONST1 = '1'	Program Memory	
Jump Address (JUMP)	4	Jump to 12-bit address 'a'	Program Memory	0≤a≤4096
Swap n	1	TOS(value) <= TOS(value)+n ; TOS(value)+n <= TOS(value)	Stack	0≤n≤127
Pick n	1	TOS(value) <= TOS(value)+n	Stack	0≤n≤127
Clear Status Register (CLRSR)	1	Status Register Bits <= '0000'	Control	Clear Status Register

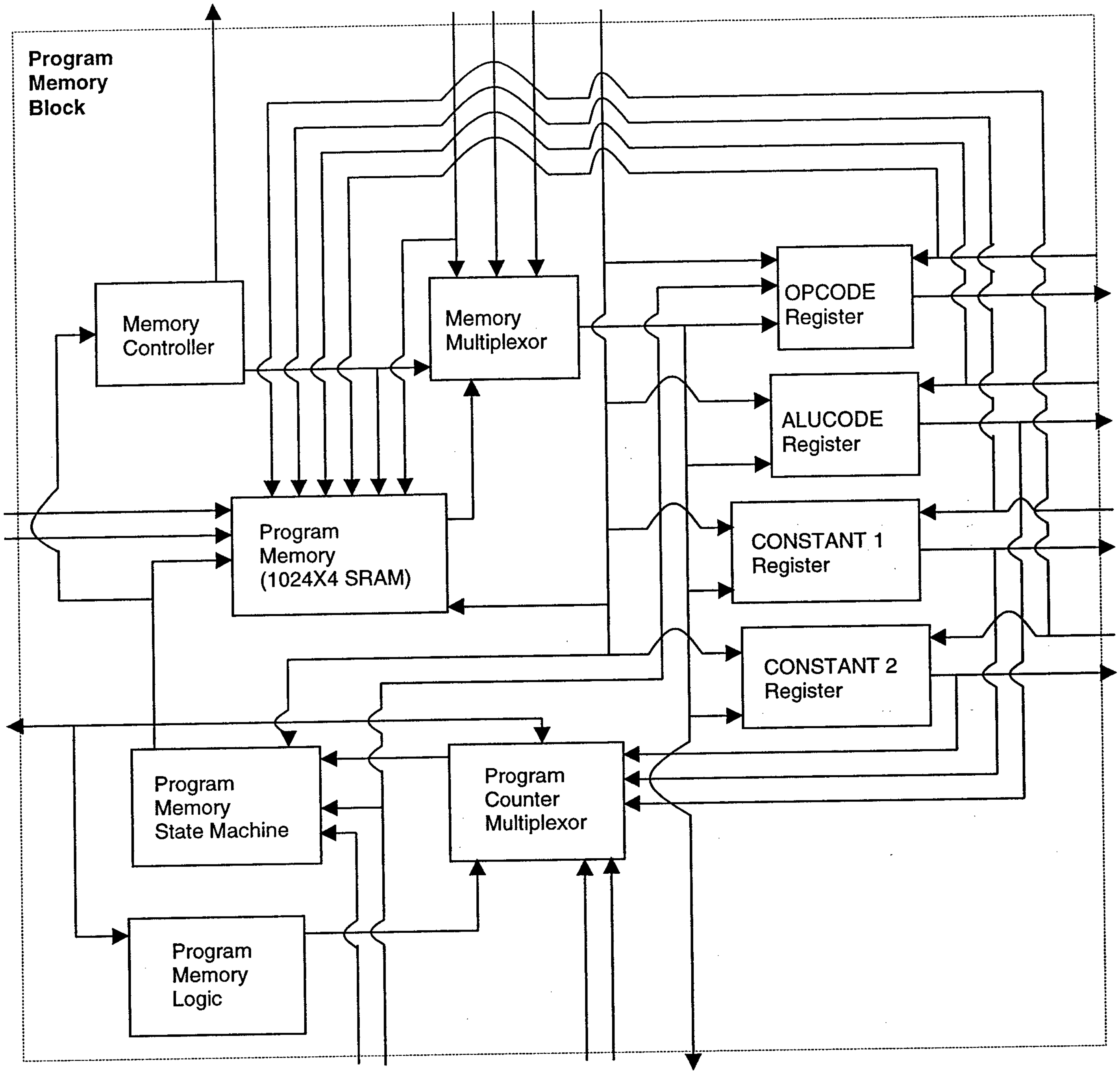


Figure 3-2. Program Memory Hierarchical Structure

address, and the current address. The Program MUX decides which of the three inputs to send through to the Program State Machine based on the values of the control lines sent from the Control block. The three elements connected together essentially are responsible for making sure the program memory address is sent to the on-chip SRAM and off-chip as the Program Memory.

3.2.1.2 Program Memory

The Program Memory is made up of seven elements that make-up a functional SRAM unit. The Program Memory includes column decode circuitry, row decode circuitry, pre-charge circuitry, pre-charge cell, write circuitry, read circuitry (sense amplifiers), and SRAM cells. Chapter Four will describe in more detail how each of these elements interact with each other in order for the SRAM unit to be read from and written to. The Program Memory gets input data from off-chip and the output is sent to the Memory Control. The Program Memory lower level hierarchical structure is shown in Figure 3-3.

3.2.1.3 Memory Control

The Memory Control is composed of two elements, which are a NOR gate (Memory Controller) and a four-line 4 to 1 multiplexor (Memory MUX). The Memory Controller controls, which SRAM either on-chip or off-chip is functional on or off. At any one time only one of the two SRAM units will be functioning as the Microdot's program memory. If the off-chip SRAM is enabled then the on-chip SRAM is disabled and vice-versa. The Memory MUX takes in three 4-bit data lines. One set of data lines is from the on-chip SRAM and another one is from the off-chip SRAM. The last set of data lines is the data input lines, which are connected to an outside source. Two of the four selections for the multiplexor are set to select the data input lines. These data input lines can be used to test and override the program memory instructions coming from either the on-chip or off-chip SRAM units. This feature is another built-in-for-test feature that will aid in determining working elements under the event that the on-chip and/or off-chip SRAM units do not function. No matter which data is selected by the Memory MUX, the output is sent to the Instruction Register.

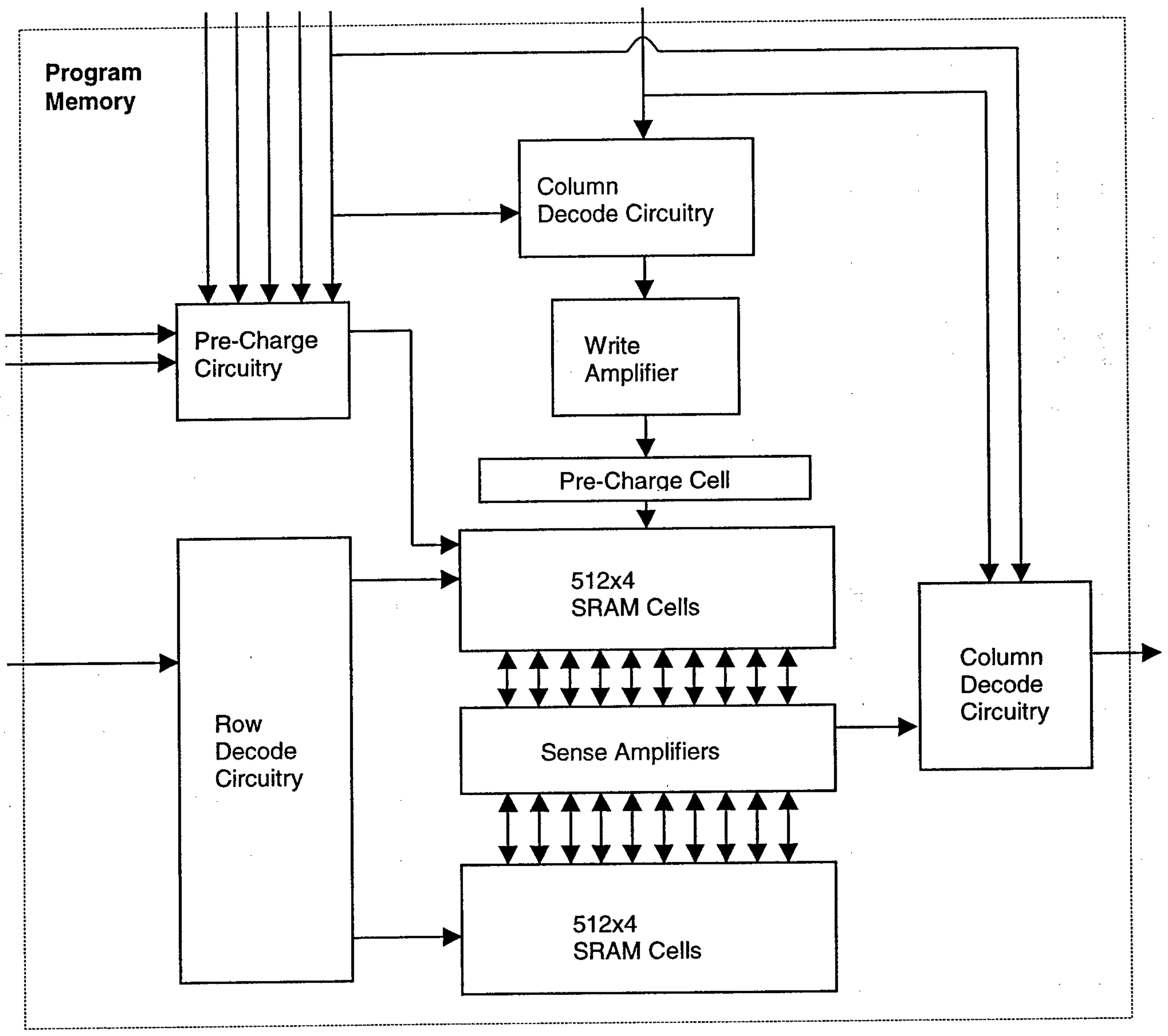


Figure 3-3. Program Memory Structure

3.2.1.4 Instruction Register

The Instruction Register is made up of four individual 4-bit registers that send out data from the Memory MUX. The four registers are responsible for transferring 4-bits of data to numerous other elements on the Microdot. The first 4-bit register is the OPCODE register, which sends a 4-bit operational code to the Control, ALU, and the Status Multiplexor blocks. The operational code tells the Microdot what instruction it will be running and thus helps the control block decide which elements will need to be operated. The next 4-bit register is the ALUCODE register, which sends a 4-bit data set to the ALU block and Control Block. The ALU code tells the Microdot, which ALU operation to perform and which control signals are needed. The two remaining registers, CONSTANT1 and CONSTANT2 registers are used for 4-bit variables for certain instructions. For example, the JUMP instruction requires a 12-bit memory address sent to the Program Counter Multiplexor. The CONSTANT1, CONSTANT2, and ALUCODE register make up the 12-bit memory address. The CONSTANT1 register is used to send data to the I/O and Stack blocks. The CONSTANT2 register is just used to create the 12-bit memory address for the JUMP instruction. All the data from the Instruction Register is sent to another block, except for the JUMP instruction scenario.

3.2.2 Stack Block

The stack block contains 10 of the 45 total elements for the Microdot. The hierarchical structure of the stack block is shown in Figure 3-4.

3.2.2.1 Stack Counter

The Stack Counter is made up of four elements, which together provide the function of generating the stack memory address that is read from or written to in the stack. The four elements are a 7-bit register (Stack State Machine), 7-bit adder (Stack Addresser), a seven line 2 to 1 multiplexor (Stack MUX), and a 7-bit incrementor/decrementor (Stack Logic). The Stack State Machine takes in the output of

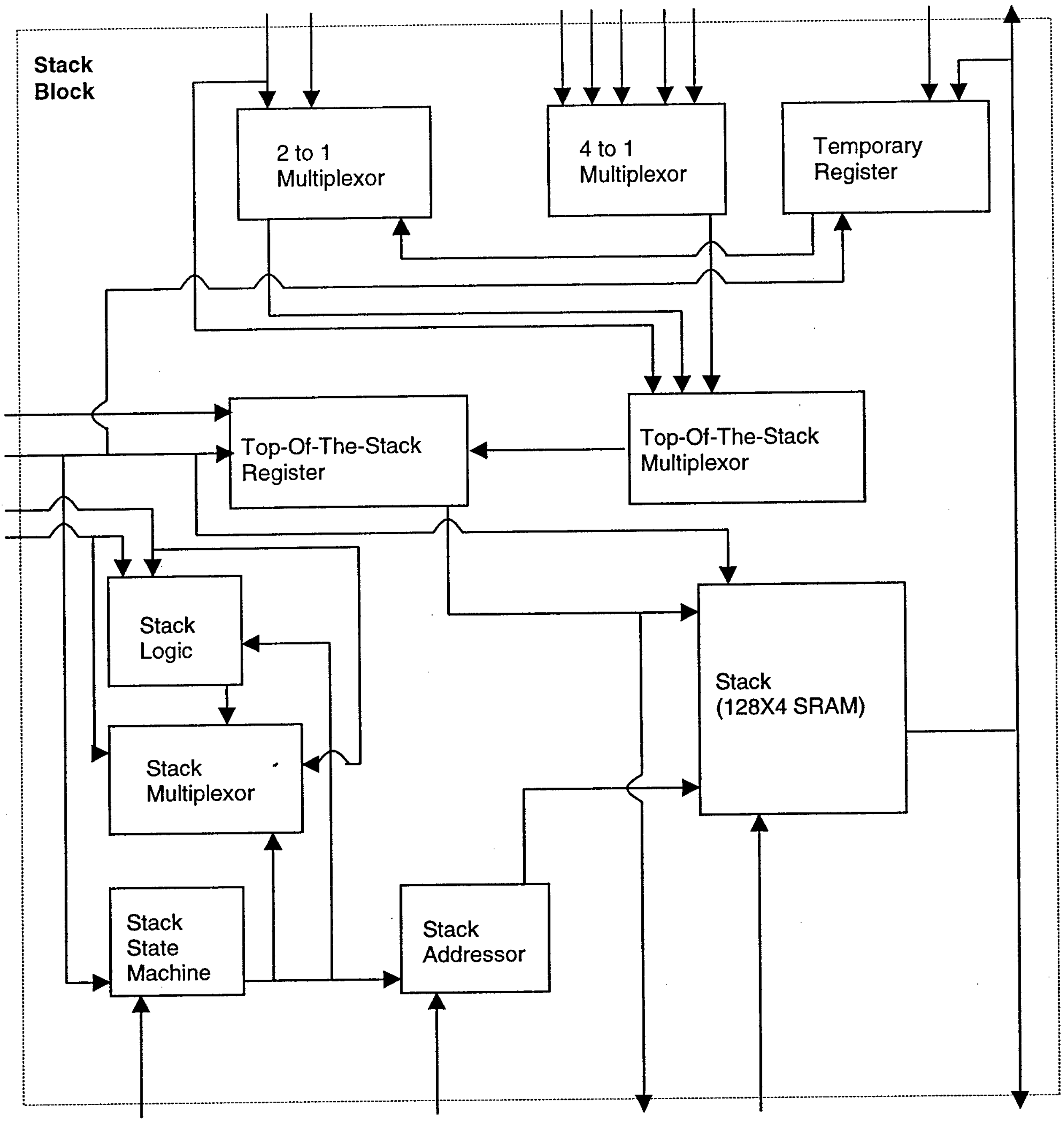


Figure 3-4. Stack Hierarchical Structure

the Stack MUX and sends the latched output to the Stack Addresser. The Stack Logic depending on control signal values either will increment or decrement the address sent out from the Stack State Machine. The output from the Stack Logic is one set of seven inputs to the Stack MUX. The Stack Addresser takes in the output from the Stack State

Machine and outputs from the Control Block. The output of the Stack Addresser is the address that is sent to the stack. The setup is designed for the incrementing, decrementing, and keeping the stack memory address static. In addition, the Stack Addresser makes it possible to reach any of the 128 stack memory addresses within the stack, because it adds two 7-bit operands. One operand comes from the Stack State Machine and the other operand comes from the ALUCODE and CONSTANT1 registers located in the Program Memory block. The output of the Stack Counter goes to the on-chip SRAM unit, known as the Stack.

3.2.2.2 Stack

The Stack is comprised of seven elements that make the Stack able to be read from and written to from any of the 128 stack addresses. The elements are the write decoder, row decoder, column decoder, pre-charge circuitry, pre-charge cell, write amplifier, and the SRAM cell. All of these elements are similar to the elements displayed in Figure 3-3 in the Program Memory Block section, which takes on the same function. Both elements are SRAM units and the only difference between them is that the Program Memory contains the Microdot's instructions, while the Stack contains actual data either produced by the Microdot or taken in from off-chip. Whether the Stack is reading from an SRAM cell or writing to one is determined by the control signals from the Control block. The output of the Stack is sent to the Status Multiplexor block and to the ALU block for data manipulation. The Stack takes in data from the TOS Cache and writes the data into the SRAM cell, which corresponds, to the memory address given by the Stack Counter.

3.2.2.3 Top-Of-The-Stack (TOS) Cache

The TOS Cache is comprised of five elements, which take in different data lines and decide which set to send through to the stack and to the Arithmetic Logic Unit (ALU) Block. The five elements are a four line 4 to 1 multiplexor (4 to 1 MUX), four line 2 to 1 multiplexor (2 to 1 MUX), another four line 2 to 1 multiplexor (TOS MUX), a 4-bit register (Temp Register), and a 4-bit register (TOS Register). The 4 to 1 MUX and the 2 to 1 MUX take in different 4-bit data lines and depending on control signal values decide which set to send through to the TOS MUX. Temp Register is a 4-bit that takes part in the key instructions of SWAP and PICK. The Temp Register sends its output to the 2 to 1 MUX. The TOS MUX is another layer of multiplexor that confirms that the correct set of data is sent to the TOS Register. The TOS Register latches through the data sent from the TOS MUX. The output of the TOS MUX is delivered to the stack as data to be written to an SRAM cell within the stack and the data is sent to the ALU Block.

3.2.3 Arithmetic Logic Unit (ALU) Block

The ALU block contains 13 of the 45 total elements for the Microdot. The purpose of the ALU block is to manipulate two 4-bit sets of data or one 4-bit set of data and determine the attributes of the results. The hierarchical structure of the ALU block is shown in Figure 3-5.

3.2.3.1 ALU Functions

There are ten functions that the ALU block can produce and these are AND, OR, Exclusive-Or (XOR), NOT, Shift Left, Shift Right, and Addition without Carry, Subtract without Carry, Addition with Carry, Subtract with Carry. There are eight elements that make it possible for the Microdot to perform these functions. The AND function is

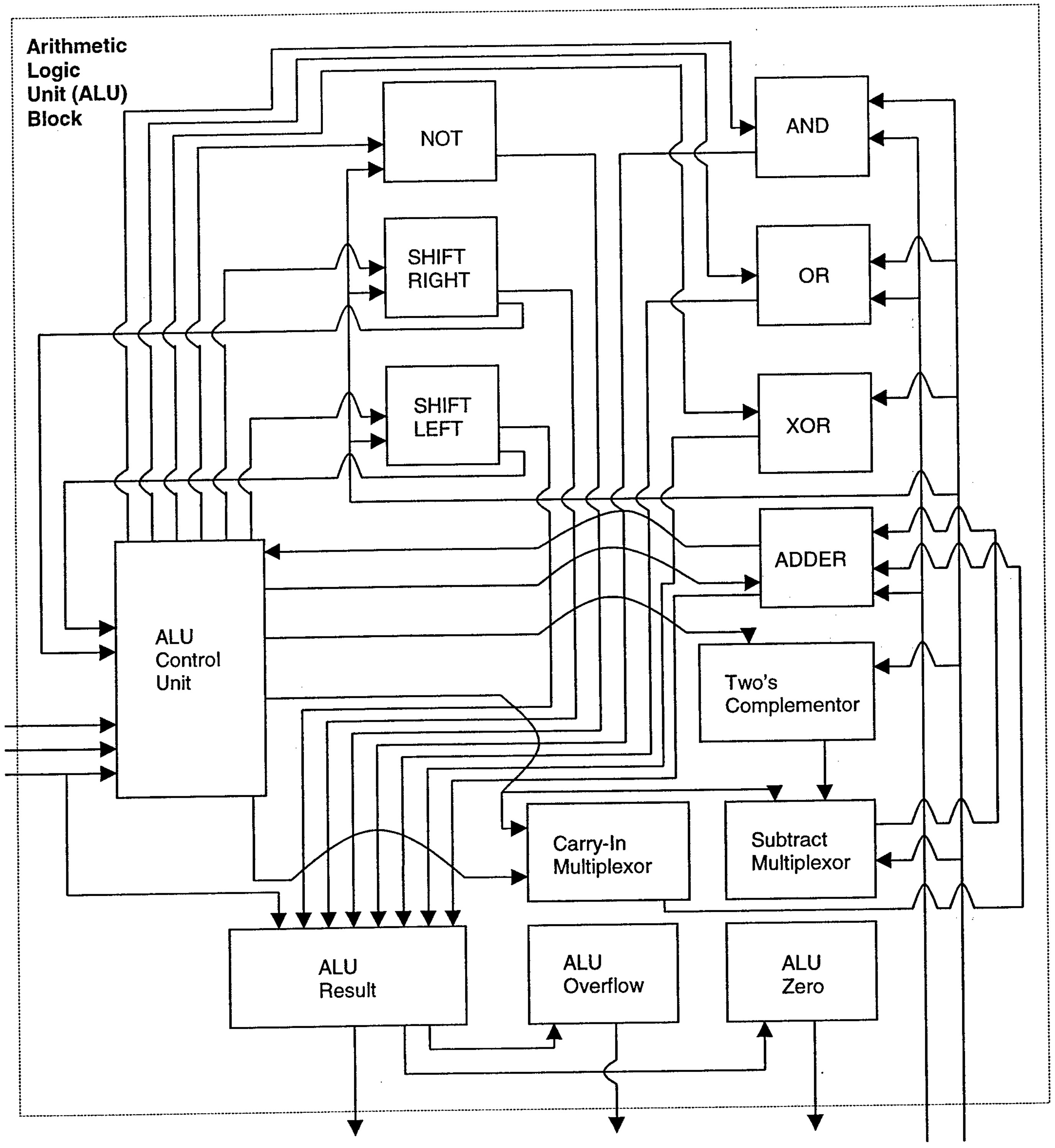


Figure 3-5. ALU Hierarchical Structure

simply the bit-wise and operation of each TOS Register output bit with each Stack output bit. The OR and XOR function follow the same process, except that each bit-wise function performing their respective gate operation on the two sets of data. The NOT function inverts the TOS Register output bits and sends the result to the ALU Result component. The Shift Left function takes the Carry bit and shifts it into the least significant bit position, while shifting the rest of the bits to the left and placing the most significant bit as the new Carry bit. The Shift Right function takes the least significant bit and shifts it out to be the new Carry bit, while the Carry bit gets shifted into the most significant bit position. The first four functions fall under the Adder/Subtractor section further on.

3.2.3.2 Adder/Subtractor

As a portion of the ALU block, the 4-bit adder/subtractor consists of four elements that make the four addition and subtraction functions possible, which are Addition without Carry, Subtraction without Carry, Addition with Carry, and Subtraction with Carry. The four elements are the Adder, Carry-In Multiplexor, Subtraction Multiplexor , and the Two's Complement that make addition/subtraction with and without using the carry bit possible. The Adder simply adds the two 4-bit operands. The Carry-In Multiplexor decides with a control signal value from the ALU Control Unit whether or not to send the Carry bit to the adder/subtractor. The Subtraction Multiplexor routes the original data or the two's complement version of the data or to the adder component. If the two's complement is sent to the adder then an subtraction operation is effectively performed, while if the original data from the stack is passed then an addition operation will occur. The Two's Complement component produces the opposite signed value of the input data and sending the result to the Subtraction Multiplexor. For example, if a negative three binary value were inputted into the Two's Complement component then the result would output a positive three binary result.

All these functions need to be controlled, so not all the functions are operating at the same time. A critical element of the ALU block, which determines the addition or subtraction of data is the ALU Control Unit.

3.2.3.3 ALU Control Unit

An ALU which contains numerous functions needs to be directed to which functions to perform at a particular time. The ALU Control Unit takes in the ALU Code and Operation Code from the Instruction Register and determines what ALU function those codes represent. ALU control signals are sent out to each functional component effectively telling which one of the elements to operate, while keeping the other elements in static operation. By not running all the ALU operations each time any instruction is passed to the ALU, the Microdot saves power by not running unnecessary operations. After the ALU Control Unit tells the elements which function to perform, the ALU Result element selects the correct output to send to the Stack and the Status Multiplexor blocks.

3.2.3.4 ALU Resultant

The ALU Resultant is made up of three elements, which are the ALU Result, ALU Overflow, and the ALU Zero. The ALU Result takes in all the functional element outputs and selects which one of the 4-bit data lines to send to the output, which is sent to the TOS Cache in the Stack block, the ALU Overflow, and the ALU Zero. The ALU Overflow determines from the output value whether to turn on or off the overflow signal, which goes to the Status Register element in the Control block. The ALU Zero element which is dependent on the ALU Result output will turn on or off the negative and zero signals, which also go to the Status Register in the Control block.

3.2.4 Input and Output (I/O) Block

The I/O block contains four of the 45 total elements for the Microdot. The hierarchical structure of the I/O block is shown in Figure 3-6.

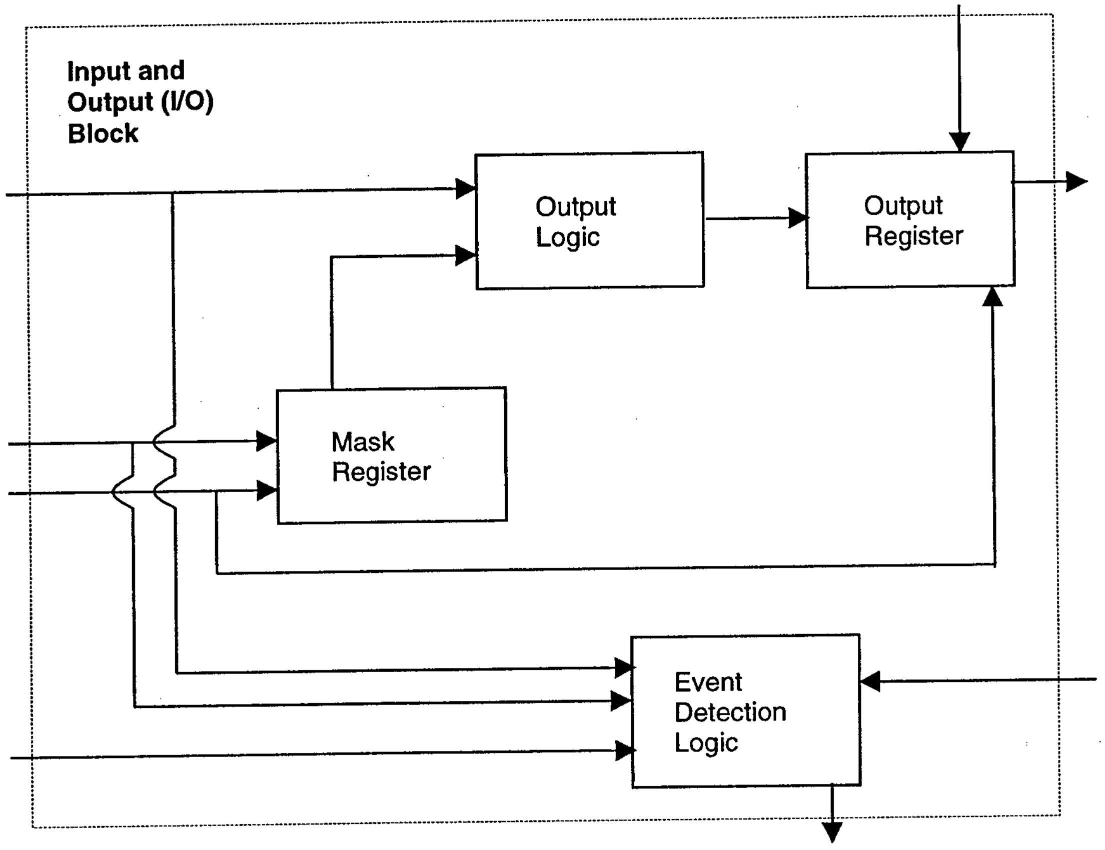


Figure 3-6. I/O Hierarchical Structure

3.2.4.1 Event Detection Logic

The Event Detection Logic simply keeps the Microdot in a waiting mode until an input value is detected that is different from the output of the TOS Register located in the Stack block. This element is used for the WAIT instruction and has constant communication with the Control Block. As soon as a different value is detected the Microdot continues on with its other instructions.

Another key function of the I/O block is determining whether the bi-directional ports should be set as inputs or outputs.

3.2.4.2 Mask Register

The Mask Register maintains values sent from the Instruction Register that determine which function each input/output port should be used for. If the bit value in the Mask Register for a port is equal to '0' then the port is set as an input port. On the contrary, if the bit value is set to '1', then the port is set to be an output port. The Mask Register will maintain loaded values until another new set is loaded in. The initial set-up values for the Mask Register for the four bits is all zeros, so the Microdot can take on inputs from the start for programming purposes. The Mask Register outputs are also sent to the Output Function.

3.2.4.3 Output Function

The Output Function is made up of two elements and they are the Output Logic and Output Register. The Output Logic takes in the Mask Register values along with the TOS Register values. If a Mask Register value is set to '1', then the TOS Register value gets passed to the Output Register. The Output Register during the STORE instruction is told to take the inputs and latch the values to the output lines of the Microdot.

3.2.5 Control Block

The Control block contains four of the 45 total elements for the Microdot. The hierarchical structure of the control block is shown in Figure 3-7.

The operation code (OPCODE) and the ALU code (ALUCODE) sent through the instruction register tell the Control Block what state to be in and what control signals to activate. A table of the operation cross-references with the specific OPCODE and ALUCODE is displayed in Table 3-2 for easy reference.

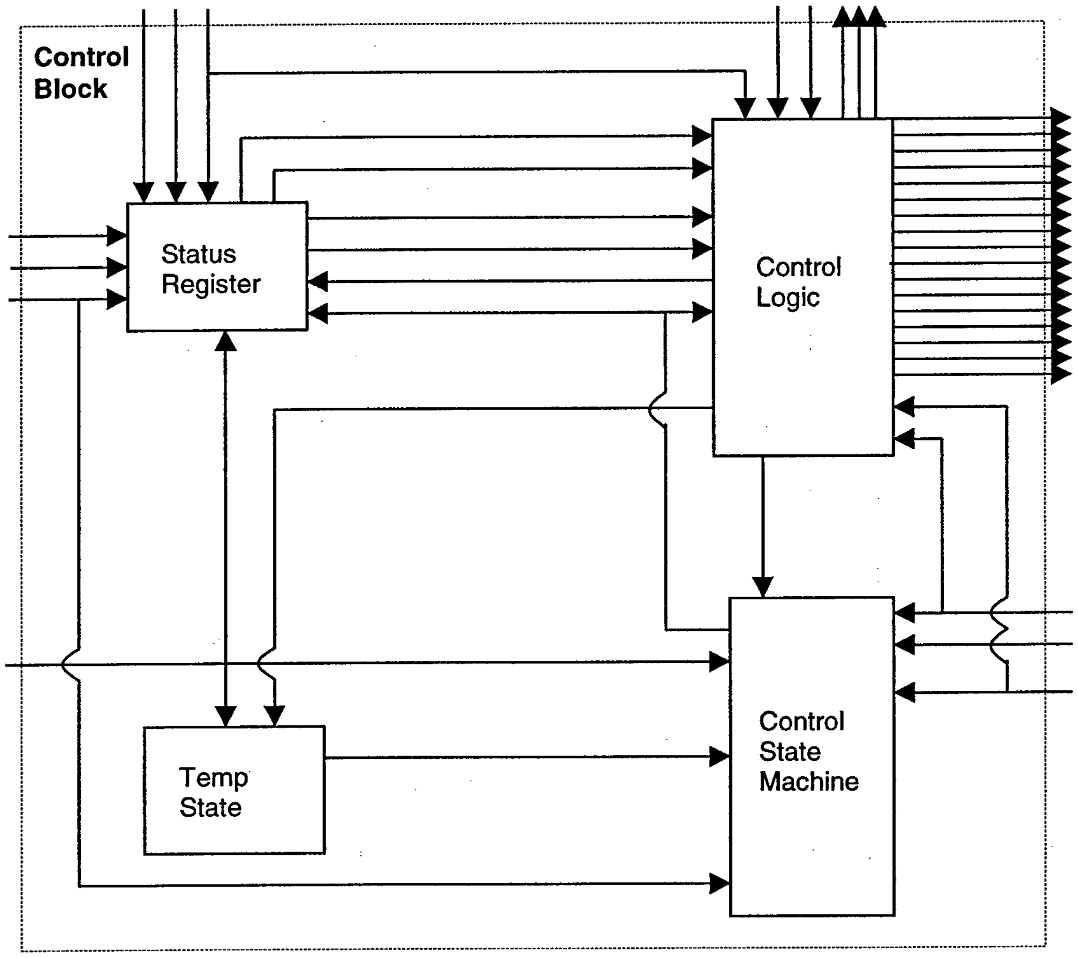


Figure 3-7. Control Hierarchical Structure

3.2.5.1 *Control Logic*

The Control Logic is responsible for taking the current state from the Control State Machine and determining which control signals to turn on or off based on the current

Table 3-2. Operation Code and Arithmetic Logic Unit Code Operation Table

Operation	OPCODE3	OPCODE2	OPCODE1	OPCODE0	ALUCODE3	ALUCODE2	ALUCODE1	ALUCODE0	Comments
Add n	0	0	0	0	0	0	0	0	Adding TOS and Stack w/o Carry
Subtract n	0	0	0	0	0	0	0	1	Subtracting TOS and Stack w/o Carry
Add w/Carry n	0	0	0	0	0	0	1	0	Adding TOS and Stack w/Carry
Subtract w/Carry n	0	0	0	0	0	0	1	1	Subtracting TOS and Stack w/Carry
And n	0	0	0	0	0	1	0	0	Anding TOS and Stack
Or n	0	0	0	0	0	1	0	1	Oring TOS and Stack
Exclusive-Or n (XOR n)	0	0	0	0	0	1	1	0	Exclusive-Oring TOS and Stack
Not n	0	0	0	1	0	1	1	1	Inverting TOS
Shift Left (SHL)	0	0	0	1	1	0	0	0	Shifting left TOS (shifting in Carry bit)
Shift Right (SHR)	0	0	0	1	1	0	0	1	Shifting right TOS (shifting in Carry bit)
Load RAM	0	0	1	0	x	x	x	x	Loading TOS register contents into Stack
Pop	0	1	0	0	x	x	x	x	Reading TOS address within the Stack
Duplicate (DUP)	0	1	0	1	x	x	x	x	Copy data from TOS to another address location within the Stack
Load	0	1	1	0	x	x	x	x	Loading the Input data into the TOS register
Store	0	1	1	1	x	x	x	x	Sending the TOS register data to the Output lines of the Microdot
Push c	1	0	0	0	x	x	x	x	Writing data onto the Stack
SetIO Mask (SETIO)	1	0	0	1	x	x	x	x	Setting the Mask bits for either inputs or outputs
Wait Mask (WAIT)	1	0	1	0	x	x	x	x	Waiting until an input line changes compared to the TOS register data
Skip Mask (SKIP)	1	0	1	1	x	x	x	x	Skips either one, two, or three instructions in the Program Memory
Jump Address (JUMP)	1	1	0	0	j3	j2	j1	j0	Jumps the Program Memory address to any location
Swap n	1	1	0	1	s3	s2	s1	s0	Swaps the TOS register data with data in any adress location within the Stack
Pick n	1	1	1	0	p3	p2	p1	p0	Picks any address location within the Stack and loads it into the TOS register
Clear Status Register (CLRSR)	1	1	1	1	x	x	x	x	Clears the Status Register data bits to all zeros

state and operational code data. The control signals go to every element in the Microdot, with the exception of the Status Multiplexor.

3.2.5.2 Control State Machine

The Control State Machine keeps track of what state the Microdot is in and determines what the next state will be. This component has been designed to follow a state diagram. The Control State Machine takes in the operational code and other key off-chip signals such as the RESET and FUNCT signals, which would be from the master microprocessor on-board the satellite. Once the off-chip master microprocessor programs the Microdot, the Control State Machine will take the loaded instructions and perform them without any monitoring or correction from the master microprocessor. At anytime, the master microprocessor can decide to reprogram the Microdot with different instructions.

3.2.5.3 Status Register

The Status Register keeps track of the Carry, Overflow, Negative, and Zero Bits. These bits help determine what kind of result came from the ALU Result and subsequently from the ALU Block. The Microdot during certain situations will use these data bits of information to decide what next action to perform. These four bits are only set during certain instructions and for some instructions a few of them may change but the others will remain constant.

3.2.5.4 Temp State

The Temp State component is used to clarify a specific state transition during the JUMP instruction. This register helps define when a different state transition should occur based upon an input value from the Control Logic. This input is latched into the register and sent out to the Control State Machine.

3.2.6 Data Acknowledge Block

The data acknowledge block contains four of the 45 total components for the Microdot. The hierarchical structure of the Data Acknowledge Block is shown in Figure 3-8.

3.2.6.1 Acknowledge Off-Chip

The Acknowledge Off-Chip component delays the Data Valid signal from the master microprocessor and sends its output to the Acknowledge Multiplexor. The value of the Data Valid signal will oscillate between '0' and '1' throughout the programming of the on-chip and off-chip SRAM units. The Acknowledge Off-Chip component is the crucial piece for handshaking between the Microdot and the master microprocessor when programming the off-chip SRAM unit. The handshaking essentially tells the master microprocessor that the off-chip SRAM has received the data and is ready for a new set of data.

3.2.6.2 Acknowledge On-Chip

The Acknowledge On-Chip component delays the Data Valid signal from the master microprocessor, when the programming of the on-chip SRAM is taking place. Its output gets sent to the Acknowledge Multiplexor. This component is critical to the handshaking process between the Microdot and master microprocessor, when on-chip SRAM is being written to.

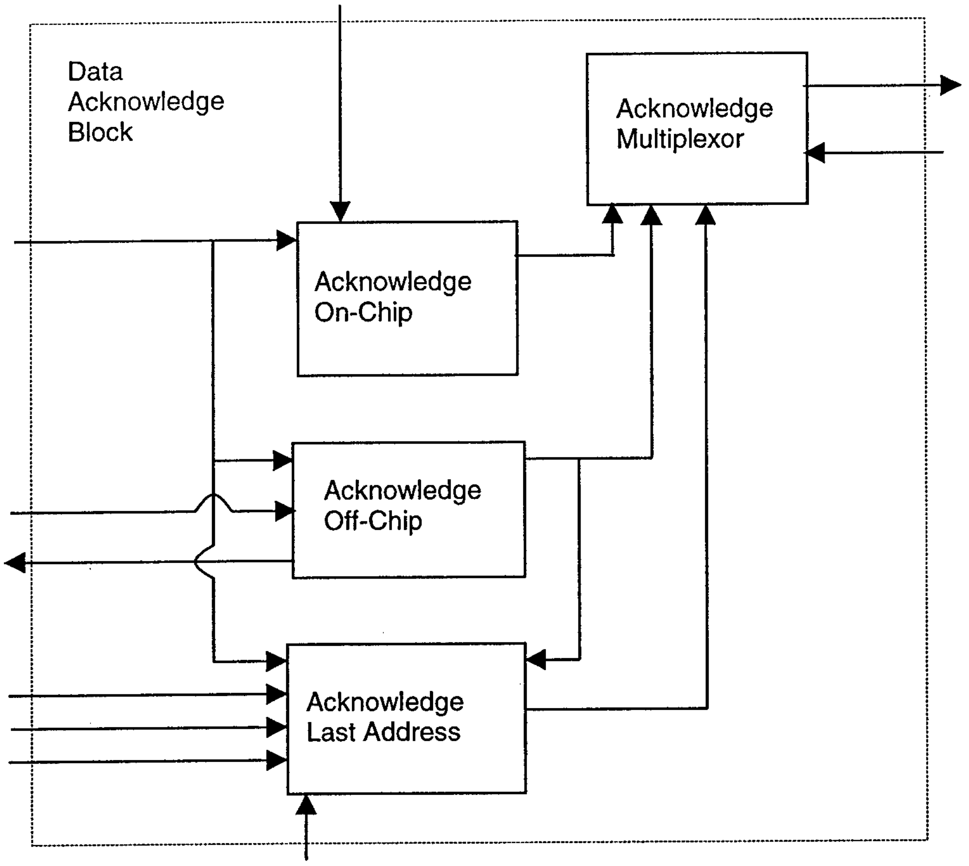


Figure 3-8. Data Acknowledge Hierarchical Structure

3.2.6.3 Acknowledge Last Address

The Acknowledge Last Address component is set up to send a constant '1' back to the master microprocessor when the address 4096 has been written to. Even if the master microprocessor sends back a '0' on the Data Valid signal line, the Acknowledge signal will remain a '1' until the FUNCT signal is set to a '0'. Thus, only when the programming function is stopped by the master microprocessor will the Microdot proceed with running the instructions that were just programmed. The Acknowledge Last Address signal is sent to the Acknowledge Multiplexor.

3.2.6.4 Acknowledge Multiplexor

The Acknowledge Multiplexor takes in the Acknowledge Off-Chip signal, the Acknowledge On-Chip signal, and the Acknowledge Last Address signal. The Acknowledge Multiplexor selects which signal to send to the master microprocessor based on which SRAM chip is enabled and whether the Acknowledge Last Address signal is high or low.

3.2.7 Status Multiplexor

The Status Multiplexor block is in a class of its own because it exists for no specific Microdot purpose, except for the designer to see the internal signal values throughout the running of all the instructions. The select lines are controlled off-chip by myself, so I can select five different data sets to view on five output pins of the Microdot. I can view the data going into the Instruction Register, the TOS Register outputs, Stack outputs, Control State Machine outputs (Current State of the Microdot), and the ALU Result outputs. This component allows me to view key signal lines in all the major areas of the Microdot. This is crucial to deducing the cause of errors, if any shall happen during the testing of the Microdot.

3.3 Design Decisions

The design goal going into the Microdot design was to consume on the order of microwatts of power during normal operation. Unfortunately, due to the size of Static Random Access Memory (SRAM) needed on chip this goal was not achieved. In addition, a radiation-hardened design was planned for the Microdot. However, due to the 0.35 sub-micron Taiwan Semiconductor Manufacturing Corporation (TSMC) design rules the design library could not contain 90-degree angle transistors. This was critical because the annular transistor contains 90-degree angles and thus I was advised not to proceed with the radiation-hardened design. There was an easily attainable clock speed requirement of operation in the kilohertz range. I believe the Microdot, as I have designed it, would be able to be run at approximately 10 MHz for a maximum clock speed. The goal for the Microdot size was to keep the area as small as possible, so I ended up keeping the size to less than 5 square millimeters. Chapter Four will go into signal line connections between elements and other blocks, along with more specific design decisions.

4. *Design Implementation*

4.1 *Microdot*

Chapter Three applied the theories discussed in Chapter Two in order to develop a working design. The overall functionality and purpose of the Microdot was explained. The instruction set of the Microdot was explained along with data flow throughout the Microdot. An overview of the Microdot design including the architecture of the top-level design and its major elements were the primary focus of Chapter Three. Interfaces between the elements within each block were diagrammed. The hierarchical structure was broken down and explained in great detail. Chapter Three gave the reader a higher-level construction of the Microdot.

Chapter Four explores the higher level design overview presented in Chapter Three at the elemental level. Chapter Four is a presentation of the design implementation of each functional block. Each block and its respective elements introduced in Chapter Three are now revisited in detail. Pin labels of each element within the different blocks are displayed. How the different blocks interact with each other is described in detail. The final design of each block, as well as other possible designs, are presented.

The Microdot is designed to complete a specific action within each state designed into the Control State Machine. Each state lasts one clock cycle and the cycle period minimum depends on the state, which takes the longest amount of time to be complete its action. Writing to off-chip SRAM is the longest action done within a state, and therefore determines the maximum operating speed for the Microdot. However, since writing to the SRAM program memory is usually done in the beginning of start-up it may be possible to run the Microdot faster when instructions are being run from the on-chip SRAM unit. The on-chip SRAM unit or Program Memory is capable of storing 1024 4-bit words and the off-chip SRAM unit is responsible for storing an additional 3072 4-bit words. This gives the Microdot the capacity to store and use 4096 4-bit instructions, the limit to the program length that can be run by the Microdot. The on-chip data Stack can store 128 4-bit words of data. This data can be manipulated by the Arithmetic Logic Unit Block and sent out through a 4-bit bi-directional port to a master microprocessor. The master microprocessor or some off-chip state machine is responsible for programming the Microdot and deciding when the Microdot can start running the programmed instructions. Once the Microdot begins running, it can run the program indefinitely. The only time the program would be interrupted would be if the master microprocessor or some other off-chip device would want to reprogram the Microdot. By simply turning on the FUNCT signal, a new program is written into the on-chip and off-chip SRAM units. The current program in the memory units would be halted and overwritten during the reprogramming process. The timing diagrams for the on-chip SRAM (Program Memory) and the Stack are located in Appendix C.

4.2 Memory Architecture

The memory architecture of the Static Random Access Memory (SRAM) units, such as the Program Memory and the Stack, are the crucial elements to the Microdot. Since the Program Memory is the largest element in the Microdot, the power consumption goal rests largely on the SRAM units. Therefore, each piece that makes up the Program Memory and the Stack will be discussed. The design for the SRAM units was gleaned from Kranz [23] and SanGregory's [24] research.

4.2.1 Two-Port Static Memory Cell

Static memory cell operation is detailed in any CMOS book, such as [4]. The two-port static memory cell can be simultaneously read from and/or written to. When a WORD line, shown in Figure 1, is asserted, the cross-coupled inverters become electrically connected to the complementary bit lines. The state of the bit lines at this point determines whether a read or write takes place. If data is actively driven onto the bit and bit-bar lines by the write amplifiers, the driven data will overcome whatever data is stored in the memory cell effectively writing to the memory cell. Otherwise, if the bit and bit-bar lines are floating in a precharged state, the existing memory cell data will be transferred onto the bit and bit-bar lines by pulling one line down to a logic-zero state and the other up to a logic-one. Depending on which line is pulled down during the read operation, the data value out will be either a logic-one or zero as stored in the memory cell. Finally, the sense amplifier will interpret the voltage change on the lines and drive SRAM output lines to the correct output value [23].

4.2.2 Precharge Circuit

The precharge cells are used to stabilize the BIT and BIT-BAR lines between memory read operations. These cells equally charge up the BIT and BIT-BAR lines, so there is an identical voltage reading on each line. Before a memory read operation begins, the state of the BIT and BIT-BAR lines is very important, because these lines are connected to a sense amplifier. The importance is due to the fact that the sense amplifier is a differential amplifier, which detects a difference in voltages between the two lines. The precharge cell, shown in Figure 4-1, is made of three N-channel transistors: two for precharging (M1 and M2) and one for stabilizing (M3) [23].

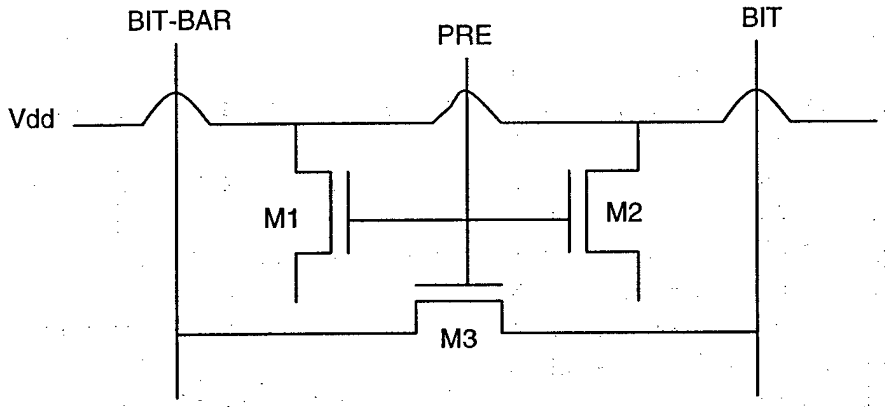


Figure 4-1. Precharge Circuit from [23]

When the PRE signal line is asserted high, all the transistors in the precharge cell are turned on. In this state, M1 and M2 bring BIT and BIT-BAR lines high to approximately 2.3 V. To guarantee that BIT and BIT-BAR lines are at the same potential, M3 shunts the two rails together. M3 allows charge sharing between the rail lines, which increases the speed of the precharging. This operation is critical to keeping valid data in the memory cells and stabilizing the differential amplifier between memory reads by forcing an equilibrium state between the BIT and BIT-BAR lines.

4.2.3 Memory Row Decoder

The Memory Row Decoder is made up of NAND gates along with inverters. In large memory structures, it has been shown that NAND gates, used as decoders, consume less power than NOR gates; however they are slower [24]. Depending on the number of rows, the number of NAND gates can vary. However, simple address buses usually using the highest significant bits and their inverted states make up the decode logic. An example of the row decode table for the Stack will help explain the set up. Table 4-1 shows the different combinations of address yielding different rows being selected.

Table 4-1. Stack Row Decoder Table

Row Selected	ADDRESS BIT 6	ADDRESS BIT 5	ADDRESS BIT 4
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

The NAND gates are selected in order to save power overall in the Microdot. The transistor size ratio for the P-channel to the N-channel is around two for low dopant concentrations. However, for high dopant concentrations in sub-micron processes the ratio is between one and one and a half [3]. The Microdot contains just over a one and a half ratio with the P-channel transistors having a width of ten lambda and a length of two lambda. While, the N-channel transistors have a width of six lambda and a length of two lambda. Transistors in series are easier to layout and save layout space. The NAND gate's P-channel transistors are in series thus yielding smaller area overall per gate.

4.2.4 Memory Column Decoder

The two memory column decoders are located in two different places. The memory column decoders have the main function of asserting the correct column that the address demands. One instantiation of the memory column decoder asserts the appropriate write amplifiers during a memory write operation. The second instantiation of the memory column decoder turns on the appropriate sense amplifiers of the on-chip SRAM unit and enables the tri-state buffers, which are connected to the output of the sense amplifiers. The memory column decoder is made up of NAND gates and is similarly constructed to the memory row decoder. The memory column decoder uses address buses that contain the least significant bits of the address. Each column pertains to a specific address combination, so each NAND gate's inputs are determined by what column the NAND gate controls. In order to attain the right combination of inputs, two address buses are routed along all the NAND gates. One address bus is the set of addresses sent from the Program Counter State Machine and the other address bus is the complement of the address sent from the Program Counter State Machine. Figure 4-2 shows a diagram of how the Memory Column and Row Decoders were laid out.

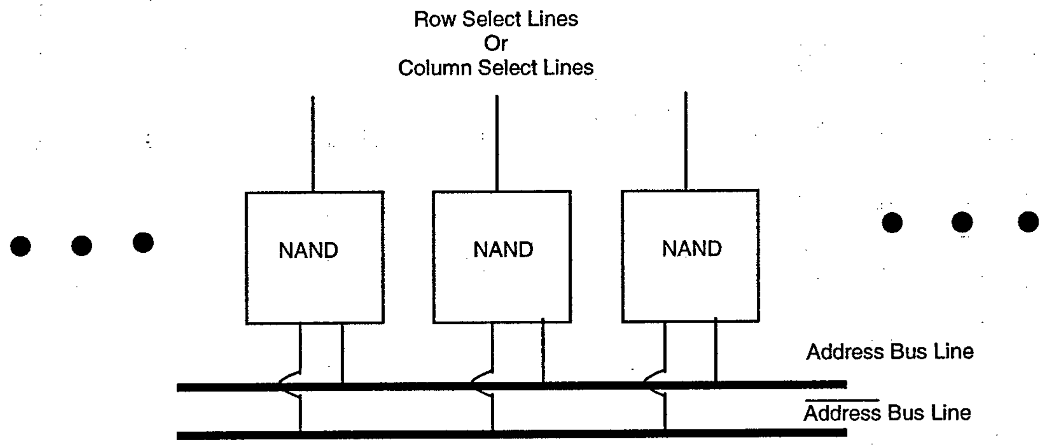


Figure 4-2. Memory Column and Row Decoder Layout

4.2.5 *Read Sense Amplifier*

The read sense amplifier detects a difference in voltage levels on the BIT and BIT-BAR lines and drives both lines to their appropriate logic state. Both lines are logically opposite one another during a memory read operation. The logic state of the BIT line is the logic value that is output from the on-chip SRAM. When a memory cell places its data on the BIT and BIT-BAR lines, the only line affected is whichever line corresponds to a logic-zero state. The other line stays at the precharge level, which is a logic-one. The rate of the drop in the line voltage is dependent on the capacitance of the line, the on resistance of the port transistor, and the size of the N-channel transistors in the storage cell. The time to fully discharge the precharge level of the line would take several nanoseconds, because of the above factors. The differential amplifier is implemented to speed up the process of detecting which line is at a logic-zero and which is at a logic-one. The differential sense amplifier is an analog device, which amplifies the difference between the BIT and BIT-BAR line values. A simple sense amplifier is used for this purpose, because of its small size and simplicity. A design choice was made to increase the transistor size in order to increase the gain, which reduces the switching time of the amplifier. By increasing transistor size an increased gain can be attained by the way of using positive feedback. The output was taken from the opposite side of the cell in order to provide the allowance for a single-stage inverter. Figure 4-3 shows how the memory sense amplifier is constructed.

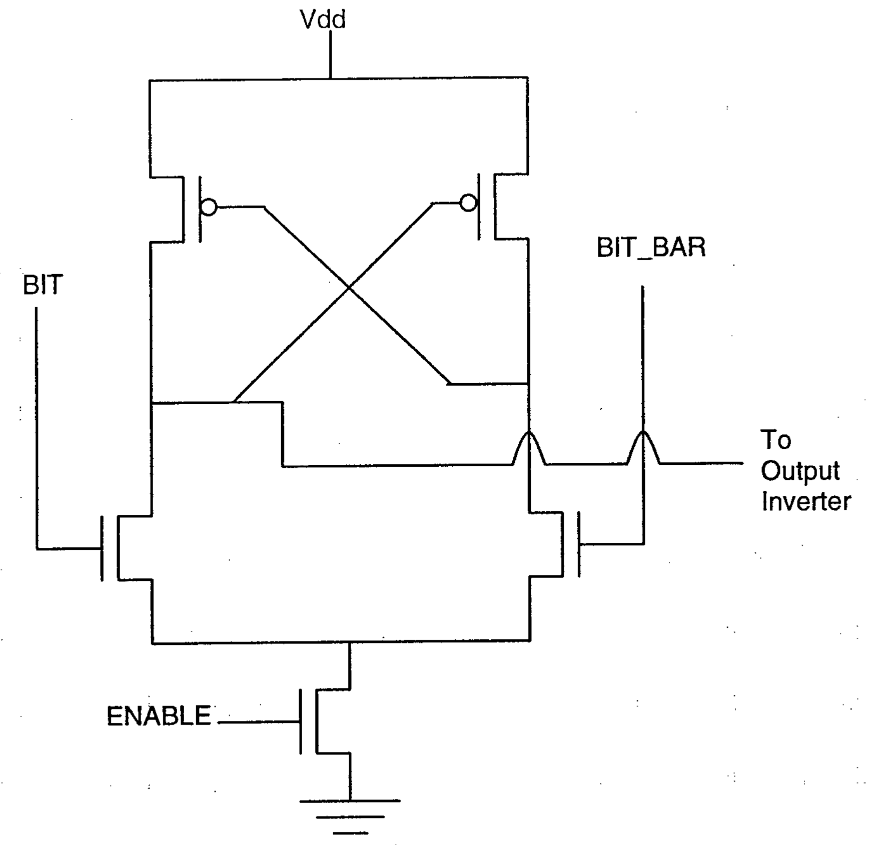


Figure 4-3. Memory Sense Amplifier Circuit from [23]

4.2.6 Write Amplifiers

The last cell needed to construct the memory was an amplifier for writing new data in to the memory cells. The write amplifier takes in data to be written to an SRAM cell and increases the drive strength of the incoming data signal. The write amplifier design is shown in Figure 4-4 and the transistor sizes are displayed by width over length dimensions in units of lambda. The design uses two stages of inverters to increase the write drive strength. A final set of inverters is enabled through a memory column decoder, which actually drive the BIT and BIT-BAR lines. One write amplifier is required for each bit column in the memory.

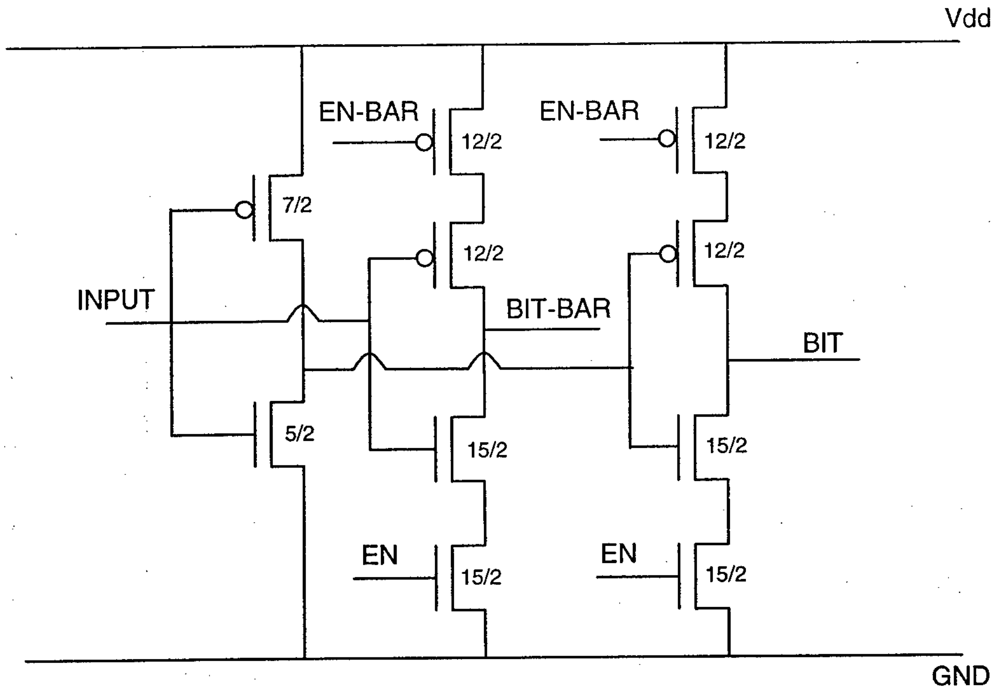


Figure 4-4. Memory Write Amplifier Circuit from [23]

The small inverter connected to the INPUT signal serves two purposes: to generate the data, and to reduce the load on the input data. Only one inverter is necessary, however such an implementation would have one inverter and the entire BIT line on the input bus. The two-inverter scheme for the BIT line helps reduce parasitic capacitance, by placing only a small inverter and two large inverters on the input bus. However, the two-inverter design does introduce an additional gate delay. The way the timing scheme occurs the data overcomes the two inverters delay before the SRAM cell is ready to be written to, so therefore the delay is transparent to the writing process [23].

4.3 Program Memory Block

The Program Memory Block, as shown in Figure 4-5, shows the signal connections for the program memory block elements. All the signal lines, either outputs or inputs, which do not have an internal connection, are connected to another block's

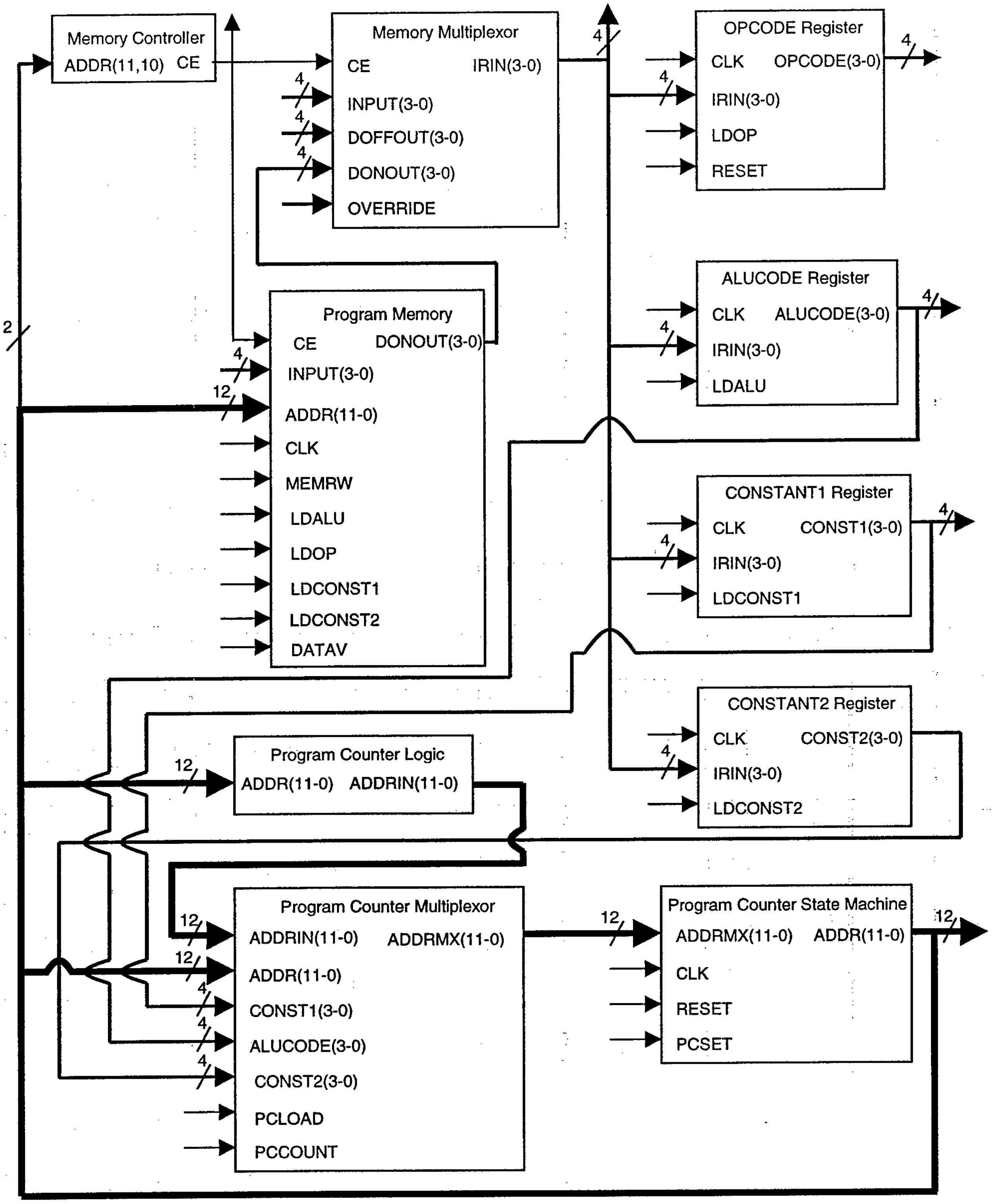


Figure 4-5. Program Memory Block Diagram

elements. The Program Memory Block is connected to 12 Microdot output pads, seven Microdot input pads, the Control Block, the Data Acknowledge Block, the I/O Block, the ALU Block, the Stack Block, and the Status Multiplexor. The SRAM unit (Program Memory) is the main element of this block. The Memory Controller enables the Program Memory through the chip enable (CE) signal. The function, number of bits, the origination, and the destination(s) of all the Microdot signals can be found in Appendix B in Table B-1. The Memory Controller simply takes the two most significant bits of the memory address and if both of them are low then the on-chip SRAM is enabled. This same enable line selects the off-chip SRAM when the chip enable value is low. The load signals (LDOP, LDALU, LDCONST1, and LDCONST2) are sent to the set of four flip-flops instruction register. These load signals are used to enable falling-edge D flip-flops within the four registers to load during the falling-edge of the clock cycle. These four load signals also have the purpose of telling the Program Memory that at the end of the clock cycle one of the four registers will be latching in the output of the SRAM unit. Therefore, the load signals trigger a read sequence to begin within the Program Memory and imply that the data on the output lines needs to be valid before the falling-edge of the clock cycle. The combined timing delays help determine clock speed limitations in the Microdot. Since multiple operations usually occur during one clock cycle, the Microdot needs to be run at the appropriate clock speeds. However, the Microdot's purpose is to be run at clock speeds in the Hertz range, since sensor readings do not have to be numerous times per second. Therefore, the fact that the Microdot runs at 20 MHz or less is not a design flaw, but intentional. The Program Memory function of either reading from or writing to SRAM cells is controlled by the Control Logic element within the Control Block through the MEMRW signal line.

The Memory Multiplexor takes in the data lines from the off-chip controlled input pins, the off-chip SRAM, and the on-chip SRAM. The Chip Enable (CE) and the OVERRIDE lines are the selectors lines to the multiplexor. Table 4-2 shows how the select lines decode which of the data lines are sent through to the four registers. If the CE line is high and the OVERRIDE line is low, then the data selected (DONOUT(3-0)) and sent to the instruction registers (OPCODE Register, ALUCODE Register, CONSTANT1 Register, and CONSTANT2 Register) is from the on-chip SRAM. If the CE line is low and the OVERRIDE line is low, then the data selected (DOFFOUT(3-0)) and sent to the instruction registers is from the off-chip SRAM. If the OVERRIDE line is high, then the data selected (INPUT(3-0)) and sent to the instruction registers is from the Microdot input lines, which is feed from off-chip.

Table 4-2. Memory Multiplexor Selection

Chip Enable (CE)	OVERRIDE	IRIN(3-0)
0	0	DONOUT(3-0)
0	1	INPUT(3-0)
1	0	DOFFOUT(3-0)
1	1	INPUT(3-0)

The select line, OVERRIDE, can bypass either SRAM unit and enter instructions into the registers externally. This feature was installed for testing purposes in case the

SRAM units fail to function properly. Instructions can be manually entered into the Microdot to bypass the SRAM and see if the other elements are working properly.

The Program Counter Logic, Program Counter Multiplexor, and the Program Counter State Machine all work together to output the correct memory address to the on-chip and off-chip SRAM units as well as to the Data Acknowledge Block. The Program Counter Multiplexor takes in the current memory address from the output of the Program Counter State Machine, the incremented by one address from the Program Counter Logic, and the jump memory address provided by three of the four registers within the instruction register. The control signals of PCCOUNT, PCLOAD, PCSET, and RESET determine the output of the Program Counter State Machine. The PCCOUNT signal tells the multiplexor to pass through the output of the Program Counter Logic, which simply takes the output of the Program Counter State Machine and increments the address by one. The Program Counter Logic automatically calculates this next address whenever the output of the Program Counter State Machine changes. However, when PCCOUNT is low and PCLOAD is high, then the jump address from the ALUCODE, CONSTANT1, and CONSTANT2 registers is passed through to the Program Counter State Machine. The three instruction registers, Program Counter State Machine, Program Counter Multiplexor, and the two control signal values (PCLOAD being high and PCCOUNT being low) allow the Microdot to jump to any 4-bit address location, which lies within either the on-chip or off-chip SRAM units. If both the PCCOUNT or PCLOAD signals are either low or high at the same time, the Program Counter Multiplexor will pass through the current address from the output of the Program Counter State Machine.

The Program Counter State Machine simply latches through the memory address the Program Counter Multiplexor gives it on the rising-edge of the clock cycle. For the latching of the memory address to work properly, the address is stabilized during the clock cycle before the new address is latched through. There are two overriding control signals that can change the normal behavior of the Program Counter State Machine. The RESET or PCSET line going high will tell the Program Counter State Machine to put all zeros on the output, thus resetting to the first memory address of the Microdot. The RESET and PCSET control lines are used independently for separate purposes. The RESET line is used when the Microdot is initializing and getting ready to be programmed. The first address written to is address zero and then the address is incremented through to 4095 or until the master processor decides to end the Microdot program. The PCSET line is used when the programming stops and the memory address needs to be reset to address zero. The starting address is key because otherwise the Microdot would start running from the last programmed address, possibly program memory address location 4095 versus address zero.

4.4 Arithmetic Logic Unit (ALU) Block

The ALU Block is composed of fourteen elements and is split into two separate figures. In Figure 4-6, the ALU Control Unit along with six of the functional elements within the ALU Block is shown. While, Figure 4-7 focuses on the elements surrounding the ADDER along with the ALU Result element.

The main focus of the ALU Block design is to accomplish the mathematical and logic operations at the lowest power for the Microdot, especially when ALU operations

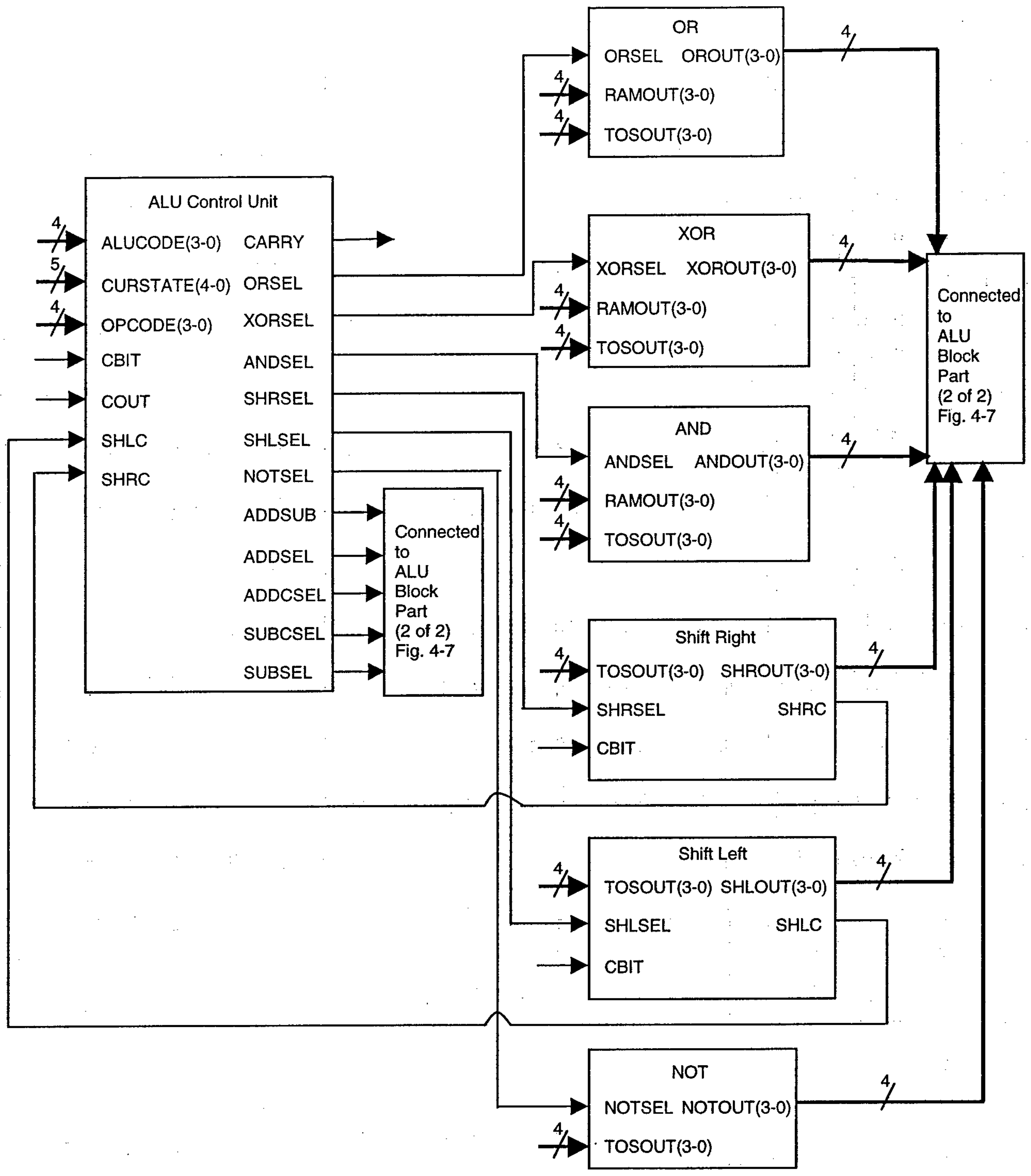


Figure 4-6. Arithmetic Logic Unit Block Diagram (1 of 2)

are not called for. The problem with ALU elements is that as long as data is changing on the input lines, the ALU elements keep processing the result. In order to keep the ALU from processing every incoming signal, I designed control signals that are sent out from the ALU Control Unit that force ALU elements to a quiescent state during non-ALU instructions. The ALU Control Unit contains seven control signals implementing the power-saving feature. Each of the functional elements (AND, OR, XOR, Shift Right, Shift Left, and NOT) contains a selector input. When the specified logic selector input is high, the element performs the operation as directed. However, when the selector input is a logic low the outputs of the elements are held constant regardless of input data changes. The ALU Control Unit is also responsible for producing the CARRY signal, which is sent to the Status Register within the Control Block. By taking the carry signals from the Shift Right, Shift Left, and the ADDER, the ALU Control Unit decides based upon the current instruction which carry bit to send out through the CARRY output line. In addition, during non-ALU operations and ALU operations that do not involve any of the carry bits the ALU Control Unit maintains the same signal whether high or low to the input of the Status Register.

An example of saving power is the logical NOT element truth table shown in Table 4-3. If the NOTSEL line is sent to a two input NAND gate along with each bit of the input data from the Top-Of-The-Stack Register, then the NOT operation is suspended by toggling the NOTSEL line.

The ALU Control Unit by taking in the ALUCODE, OPCODE, and the CURSTATE signals, can determine which one of the ALU functions needs to be running and therefore shuts the rest of the ALU functions down to save power. These kinds of power saving techniques have been applied to every functional element including the

Table 4-3. NOT Truth Table

TOSOUT	NOTSEL	OUTPUT
0	0	1
0	1	1
1	0	1
1	1	0

ADDER, OR, XOR, AND, NOT, Shift Left, and Shift Right, which are shown in Figure 4-7.

In order to operate the adder/subtractor, one needs several other elements that function along with the adder. Figure 4-7 shows the additional elements to be the Two's Complementor, Subtract Multiplexor, and the Carry-In Multiplexor. The Two's Complementor takes in the data from the Stack and depending on the ADDSUB signal will either output the two's complement of the data or will pass through all ones to the Subtract Multiplexor. Producing the two's complement only when necessary is yet another power saving technique, which only uses the element when it is needed. The Subtract Multiplexor decides which set of data to send to the ADDER. Either the two's complement data or the original data from the Stack is passed to the ADDER. The ADDSUB signal in this case is the selector line for the multiplexor. The Carry-In Multiplexor takes in control signals from the ALU Control Unit and the CBIT signal from the Status Register in the Control Block. The different select lines such as ADDSEL, ADDCSEL, SUBSEL, and SUBCSEL tell the multiplexor whether the adder will be performing its operation with the carry bit or without the carry bit. When either ADDCSEL or SUBCSEL is high, the CIN signal sends through the CBIT signal value to

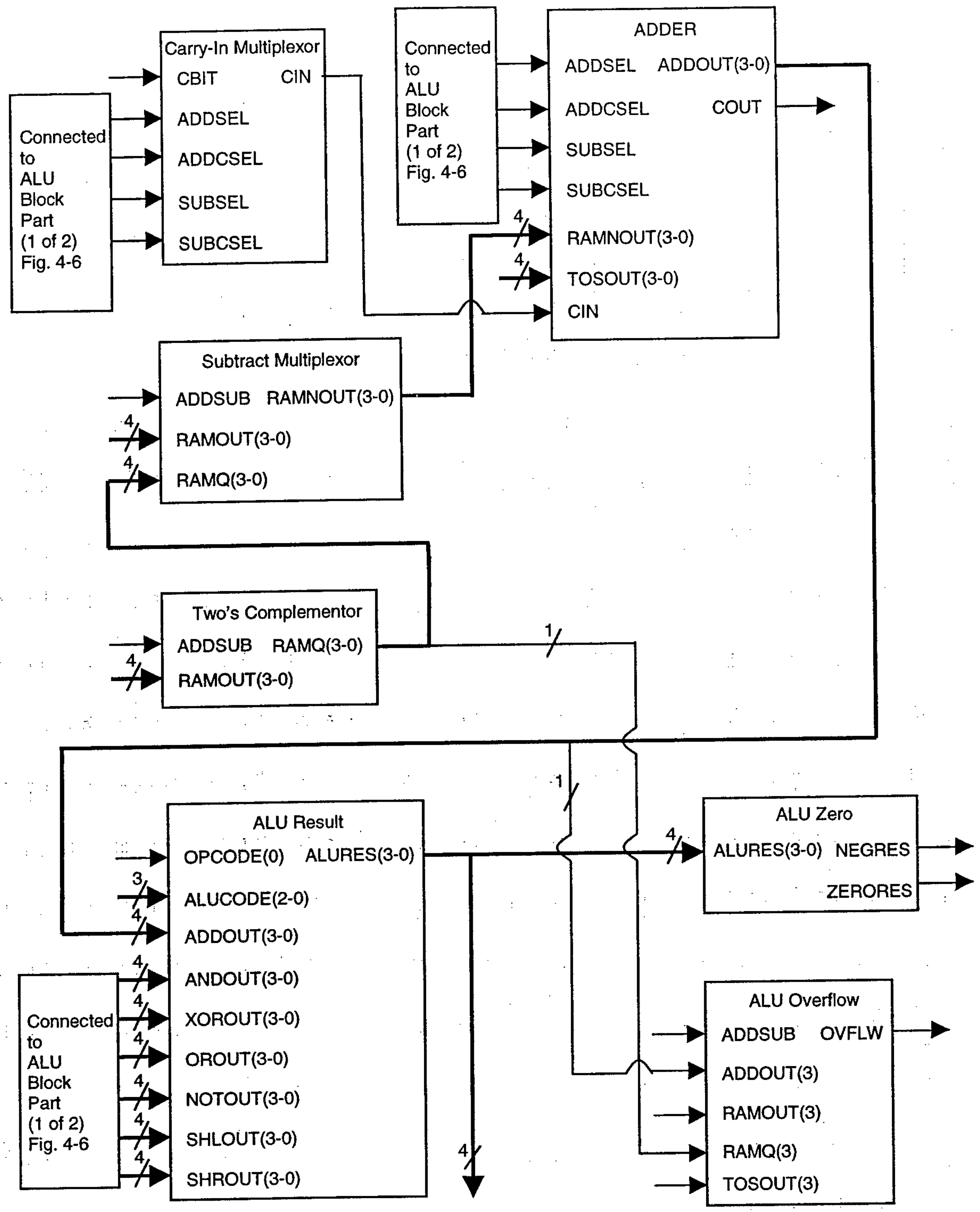


Figure 4-7. Arithmetic Logic Unit Block Diagram (2 of 2)

the ADDER element. However, if all of the select signals are low or if either ADDSEL or SUBSEL are high, then the CIN output is set to zero regardless of the CBIT signal value on the input.

The last critical element to the ALU Block is the ALU Result element, because it sends the final data outside the ALU Block to the Stack Block and the Status Multiplexor.

The ALU Result inputs are all the outputs from all the functional units and parts of the ALUCODE and the OPCODE. From the pieces of the ALUCODE and the OPCODE, the ALU Result can determine which set of data to pass to outside the ALU Block. This final set of data is sent to the ALU Zero element for determination if the result was negative or zero. The ALU Zero element sends its outputs to the Status Register. Another element that sends its output to the Status Register is the ALU Overflow element. The ALU Overflow element takes in the most significant bits from the Top-Of-The-Stack Register, ADDER, Stack, and the Two's Complementor along with the ADDSUB bit. The most significant bits along with the ADDSUB bit determine whether there was an overflow situation or not. The ALU Overflow simply determines the overflow status and reports that information back to the Status Register on the OVFLW signal line.

The result of the ALU Block is sent to the Stack Block for storage and possible output to off-chip.

4.5 *Stack Block*

The data storage unit in the Microdot is the Stack, SRAM unit, which is responsible for holding incoming data from off-chip through the bi-directional ports. The Stack along with the ALU Block can alter the data and perform various operations on either one set or two sets of 4-bit data. Elements such as the 2-to-1 Multiplexor, 4-to-1 Multiplexor, Temporary Register, Top-of-the-Stack Multiplexor, Top-of-the-Stack Register, Stack Logic, Stack State Machine, Stack Addressor, and the Stack Multiplexor make it possible for the Stack to operate properly.

The Stack Block, which is made up of 10 elements, is shown in Figure 4-8. There are four elements, which maintain and assign the memory address location for the Stack, SRAM unit. The maintaining and assigning of the Stack's memory address are important functions, because without the proper address data will be written to or read from the wrong location. The Stack memory uses a 7-bit address, which gives the Stack a total of 128 memory locations to work with. The Stack State Machine latches through data from the Stack Multiplexor on the rising-edge of the clock cycle. The output of the Stack State Machine is sent to the Stack Addressor and to the Stack Logic. The RESET signal line to the Stack State Machine when is a low or zero state sets the output to all ones. At the start up of the Microdot, the first Stack memory address is location 127, or '1111111_2' in binary code. The reason this start up address was picked minimized the number of Stack element related logic gates for the Microdot, thus saving power.

The Stack Addressor is a 7-bit adder, which takes the 7-bit output of the Stack State Machine and adds it to the 7-bit OFFSET input from the Control Block. This gives the ability to effectively jump to any of the 128 memory address locations in the Stack. The memory address normally decrements down, when data is pushed on, from address 127 to 0. In order to jump back to a previous location, the Stack Addressor simply adds the current address to the OFFSET to reach an old address. However, a reminder that the

Stack Addressor will yield the memory address or result without a carry bit, so I can add one to address 127 and roll over to the numerically lower valued address location which

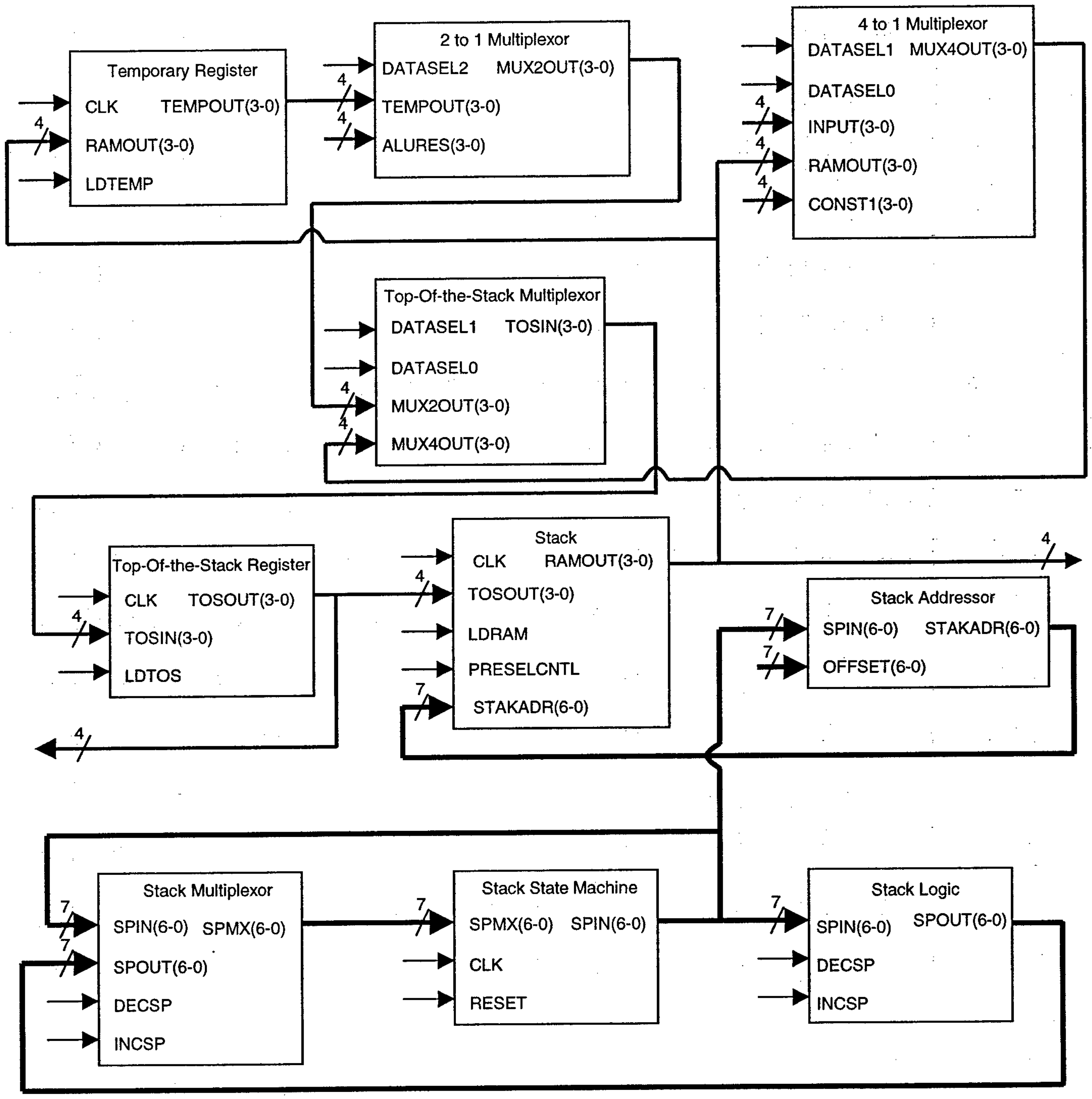


Figure 4-8. Stack Block Diagram

would be address 0. The Stack Addressor simply adds whatever is on its inputs. The inputs are two 7-bit operands (SPIN signal and OFFSET signal) from the Stack State Machine element and the Control Logic element, respectively. If the 7-bit SPIN signal or the 7-bit OFFSET signal changes, the Stack Addressor will sum the inputs and calculate a new memory address. The new memory address is sent from the Stack Addressor to the Stack. The memory address is meant to decrement down from address 127 versus incrementing up from address 0. By decrementing the Stack memory address, the addition operation will be the appropriate operation when jumping to another one of the 128 memory address locations. A jumping of Stack memory address locations occurs when a value greater than zero is asserted on the OFFSET signal line. When OFFSET is set to a value greater than zero, the current Stack memory address (SPIN signal) and the OFFSET signal are added together to create a new Stack memory address. If I chose to have the Stack memory address increment from a starting address of zero rather than decrement from a starting address of 127, then a subtraction operation would need to occur in order to complete an accurate jumping of Stack memory address locations. As a design decision, I choose to have the address decrement down from address 127 to 0 in order to save building a 7-bit two's complementor that would be connected to the Stack Addressor.

I chose to decrement the Stack memory address after pushing or writing to a memory location and increment the Stack memory address after popping or reading from the Stack. The current Stack memory address always is referred to as the Top-of-the-Stack memory address. When pushing or writing to the Stack, the data is written into the Top-of-the-Stack memory address location. At the beginning of the next clock cycle, the

Stack memory address is decremented by one. The new Stack memory address is now the Top-of-the-Stack memory address. The decrementing of the memory address sets up for the next push or write to the Stack. However, if a pop or read occurs then the Stack memory address is incremented by one. The incrementing operation makes the last set of data written to the Stack available on the output lines of the Stack. The incrementing of the Stack memory address occurs to counter the last decrement, which occurred at the end of the last push or write operation.

Stack Logic plays an important role with selecting the next Stack memory address. The 7-bit SPIN signal is input to the Stack Logic from the Stack State Machine and the Stack Logic either increments or decrements the SPIN signal by one. Whether or not to increment or decrement, depends on the control signals INCSP and DECSP from the Control Logic within the Control Block. If one control signal is high while the other is low, then the operation of the high dominates and controls the output sent to the Stack Multiplexor. However, when the control signals are both low or both high, the Stack Logic simply passes the 7-bit SPIN signal through to the Stack Multiplexor. Under the condition of the control signals having the same value, the SPIN signal is sent to the Stack Multiplexor so the Stack memory address can remain constant. A constant Stack memory address is needed when neither a push or pop operation has been called for. Also if a control signal malfunction occurs and both signals are a logic-one, a constant Stack memory address is needed to isolate the malfunction and not have the malfunction propagate to the Stack Multiplexor and eventually to the Stack.

The Stack Multiplexor takes on the output address data from the Stack State Machine and the Stack Logic. DECSP and INCSP are control selector signals the multiplexor uses to determine address to pass through to the Stack State Machine. If either the DECSP or INCSP are high then the Stack Logic data is sent through, but if neither are high or both are high then the Stack State Machine data is passed through. The DECSP and INCSP are never supposed to be high at the same time, but as a precaution, I designed in a safety feature refreshing the old memory address until the conflict is resolved by the Control Block.

There are five elements, which are responsible for delivering the correct set of data to the Stack's data input lines. The Temporary Register is the first of these five elements to be discussed. The SWAP and PICK instruction combination discussed in Chapter Three needs a data storage element to make the instruction work properly. The Temporary Register is the data storage element needed when these instructions are used together in order to access data at any of the 128 address locations. The register takes data from the Stack output and latches it through on the falling-edge of the clock cycle and while the LDTEMP signal is high. Only when the falling-edge of the clock cycle occurs and the LDTEMP signal is high will the Temporary Register latch the data through to the 2-to-1 Multiplexor element.

The 2-to-1 Multiplexor element takes in data sent from the Temporary Register and the ALU Result in the ALU Block. The Temporary Register sends the 4-bit TEMPOUT signal and the ALU Result sends the 4-bit ALURES signal. The DATASEL2 signal from the Control Logic element is the control selector signal for the 2-to-1 Multiplexor element. If the DATASEL2 signal is low then the 4-bit ALURES signal is passed through to the Top-Of-The-Stack (TOS) Multiplexor. However, if the signal is low then the 4-bit TEMPOUT signal is passed through.

The 4-to-1 Multiplexor element takes in data sent from the CONSTANT1 Register, the four bi-directional ports connected to the master microprocessor, and the output of the Stack. The fourth set of data is a set of zeros in case of a malfunction of the control selector signals (DATASEL1 and DATASEL0). The DATASEL1 and DATASEL0 control selector signals from the Control Block determine which set of data is passed to the TOS Multiplexor. If the DATASEL1 and DATASEL0 are both high, then the 4-bit RAMOUT signal is passed. If the DATASEL1 signal is high while the DATASEL0 signal is low, then the 4-bit INPUT signal is passed. If the DATASEL1 signal is low while the DATASEL0 signal is high, then the 4-bit CONST1 signal is passed. In the event that both control signals are low, then all zeros are passed to the TOS Multiplexor. Table 4-4 shows the control selector values and which corresponding set of data is passed to the output of the 4 to 1 Multiplexor element.

Table 4-4. 4 to 1 Multiplexor Truth Table

DATASEL1	DATASEL0	4 to 1 Multiplexor Output
0	0	'0000_2'
0	1	CONST1(3-0)
1	0	INPUT(3-0)
1	1	RAMOUT(3-0)

The TOS Multiplexor determines if either the output from the 2-to-1 Multiplexor or the 4-to-1 Multiplexor gets sent to the Top-Of-The-Stack (TOS) Register. The control signals DATASEL1 and DATASEL0 determine the output of the TOS multiplexor. If the DATASEL1 and DATASEL0 are both low, then the 4-bit MUX2OUT signal gets passed to the TOS Register. If any other signal combination of DATASEL1 and DATASEL0 occurs the 4-bit MUX4OUT signal gets passed to the TOS Register.

The TOS Register takes in data from the TOS Multiplexor and latches the data through, when the falling-edge of the clock cycle occurs and the LDTOS control signal is high. The output only changes when the LDTOS signal is high and the falling-edge of the clock cycle occurs, otherwise the last set of data latched through remains on the output lines (TOSOUT(3-0)). Output data from the TOS Register gets sent to the Stack, ALU Block, and the Input and Output Block.

The Stack is a 128 location by 4-bit SRAM unit that stores data for the Microdot. The 2-to-1 Multiplexor, 4-to-1 Multiplexor, Temporary Register, TOS Multiplexor, and the TOS Register mentioned above control which data gets written into the Stack through the 4-bit TOSOUT signal. The Stack is set up to read during the rising part of the clock cycle and write during the falling edge of the clock cycle. The reading occurs during the rising part of the clock cycle, because the output data needs to be manipulated by the ALU Block and latched into the TOS Register on the falling-edge of the clock cycle. Since data is latched through by the TOS Register on the falling-edge of the clock cycle, input data to the Stack is only valid two nanoseconds after the falling-edge of the clock cycle, so that is why the write occurs in the falling part of the clock cycle. When the reading and writing of data occurs is simply based off of when data needs to be available to the other elements within the Microdot. The PRESELCNTL signal turns off the pre-charge circuitry mentioned in Section 4.2.2, which is used for the read cycle. The LDRAM control signal triggers the write cycle for the Stack. Each of those control signals along with the correct part of the clock cycle triggers the read or write cycle. The output of the Stack is sent to the ALU Block, Temporary Register, 4-to-1 Multiplexor, and the Status Multiplexor. The Stack is made up of the elements mentioned in Section 4.2.

4.6 Input and Output (I/O) Block

Four bi-directional ports are located in the Microdot, which can be switched between either being input pins or output pins. There are four elements, Mask Register, Output Logic, Event Detection Logic, and Output Register, that control these bi-directional ports. These elements make up the Input and Output (I/O) Block, shown in Figure 4-9.

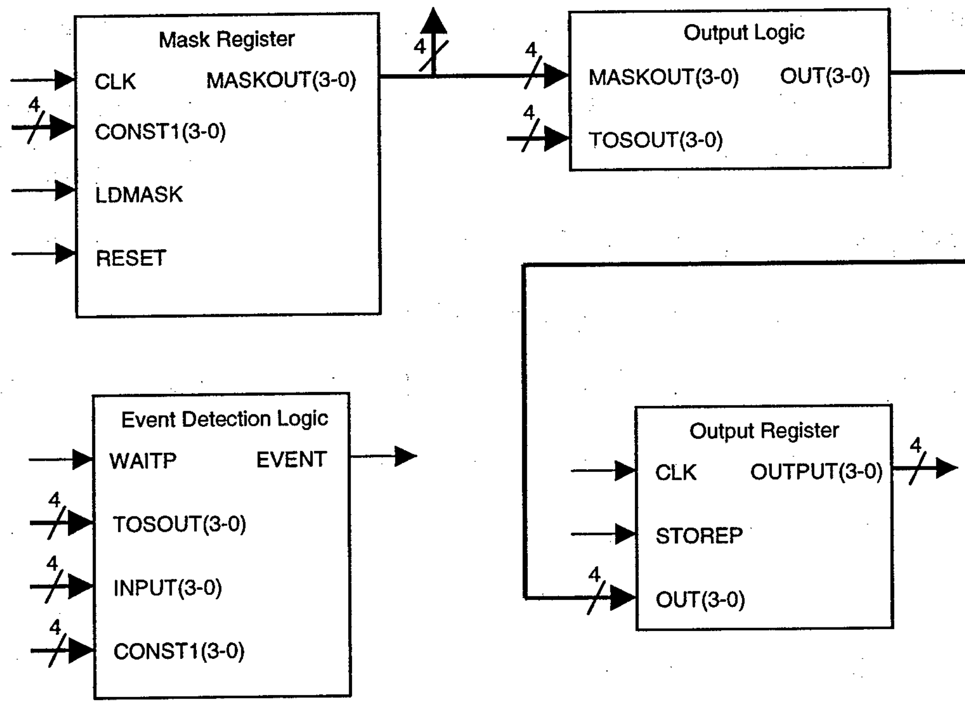


Figure 4-9. Input and Output Block Diagram

The Event Detection Logic is a separate element, which is not connected to the other I/O block elements, but relies on the TOS Register output, CONSTANT1 Register output, the bi-directional ports, and the Control Block. Event Detection Logic plays an important role in performing the WAIT instruction for the Microdot. Each bit of the 4-bit CONST1 signal sets up which input bit to compare. If the bits from CONSTANT1 register are high then the TOS Register output bit will be compared to the input bit on a bit-wise basis. If however, a bit from the CONSTANT1 Register is low, then the respective bits from the TOS Register output and the bi-directional ports are not compared. Event Detection Logic takes in the TOS Register output and compares the values to the input lines in the bi-directional ports. If any of the bit-wise comparisons are different and they have been targeted to be compared, the EVENT signal goes high and is sent to the Control Block. Therefore, the Event Detection Logic has the ability to compare all four sets of bits from the TOS Register and the bi-directional ports. Depending on the values from the CONSTANT1 Register, the Event Detection Logic can be set to compare four, three, two, one, or zero bits from the TOS Register and the bi-directional ports. This gives the option of waiting for a specific input bit to change or even a whole 4-bit set of data to change before the Microdot continues on with other instructions.

The Mask Register sets the bi-directional data ports to be either inputs or outputs. The 4-bit MASKOUT signal sends a single bit to each of the four bi-directional ports, which are connected to the master microprocessor and possibly sensor, external testing device, etc. If the bit output from the Mask Register is high then the bi-directional is set to be an output. If the bit output is low, then the bi-directional port is set to be an input. The Mask Register is loaded on the falling-edge of the clock when the LDMASK signal is high. It is important to notice that the bi-directional ports will be set as inputs from start-up with the RESET signal being low. The Mask Register must be loaded in order to change the direction of the ports from this initial set up. The outputs of the Mask Register are sent to the Output Logic and the bi-directional ports.

The Output Logic takes in the Mask Register outputs and the TOS Register outputs. If a bit from the Mask Register is set to low, then the corresponding TOS Register bit is sent to the Output Register. However, if a Mask Register bit is high, then the bi-directional port is set to be an input, thus the output of Output Logic will not be sent off-chip. The output of the Output Logic is always sent to the Output Register whether the bi-directional port is set up to be an input or an output.

The Output Register latches the output from the Output Logic on the falling-edge of the clock and the STOREP control signal is high. The output of the Output Register is sent to the output lines of the bi-directional ports, which are separate from the input lines of the bi-directional ports.

4.7 Control Block

The Control Block is the main control unit, which controls every element within the Microdot, except for the Status Multiplexor. The four elements play a critical role in sending control signals at the correct time to the elements to implement their specific function.

Shown in Figure 4-10, is Control Block Diagram indicating all the interconnections between the four elements. The Status Register stores the carry, negative, overflow, and the zero bits. These bits are sent to the Control Logic, to help determine control signal values. The output bits only change during ALU instructions

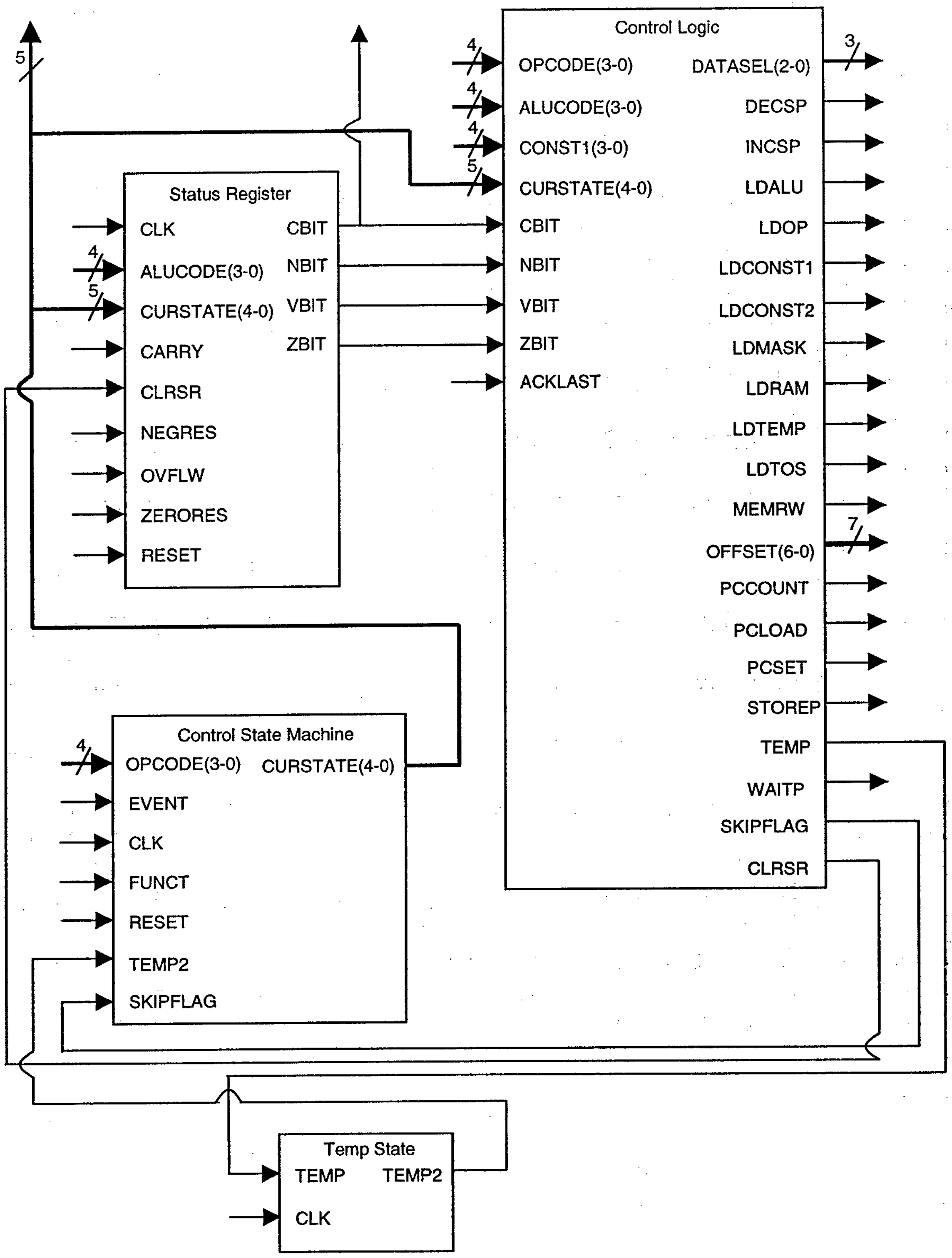


Figure 4-10. Control Block Diagram

and that is why the 5-bit CURSTATE and 4-bit ALUCODE signals are inputs to the Status Register. The CLRSR control signal comes from the Control Logic and is used to set the output bits to all zeros when asserted. At start-up when the RESET signal is asserted low, the Status Register begins with zeros on the output bits.

The Status Register latches in values from the ALU Block on the falling-edge of the clock when the CURSTATE signal and ALUCODE signal values are appropriate. During the AND, XOR, OR, NOT, Shift Left, and Shift Right instructions not all of the status bits are set to whatever the input values are. During these instructions the overflow bit is not set, because there is no possibility for an overflow condition to occur. The carry bit is not set during the AND, OR, XOR, and NOT instructions, because these are bit-wise operations. Table 4-5 shows what outputs are set during which instructions.

Table 4-5. Carry, Negative, Overflow, and Zero Bit Setting

Instruction	Carry Bit	Negative Bit	Overflow Bit	Zero Bit
Add n	X	X	X	X
Subtract n	X	X	X	X
Add n w/carry	X	X	X	X
Subtract n w/carry	X	X	X	X
AND n		X		X
OR n		X		X
XOR n		X		X
NOT n		X		X
Shift Left	X	X		X
Shift Right	X	X		X

The Control Logic takes the data from three registers from the Instruction Register to determine control signal values during particular instruction sequences. The ACKLAST input signal comes from the Data Acknowledge Block and tells the Control Logic when to turn the MEMRW control signal from a logic-zero to a logic-one. The

ACKLAST signal is used to stop writing data to memory address locations on the off-chip SRAM or on-chip SRAM after the last memory address location 4096 has been written to. When programming the Microdot, there is only 4096 memory address locations that can be filled with data. An internal trigger turns off the programming when the 4096th memory address location is reached. The internal trigger, ACKLAST signal, prohibits the writing over of previously programmed memory address locations. The status bits sent to the Control Logic similar to the OPCODE and ALUCODE signals help determine control signal values during the SKIP instruction. The status bits help determine whether the SKIPFLAG signal from the Control Logic element should a logic-one or logic-zero during the SKIP and SKPT states. If any of the status bits are a logic-one and any of the CONST1 signal bits are a logic-one during the SKIP and SKPT states, the SKIPFLAG signal is asserted high. Table 4-6 shows the control signals activated during each instruction of the Microdot.

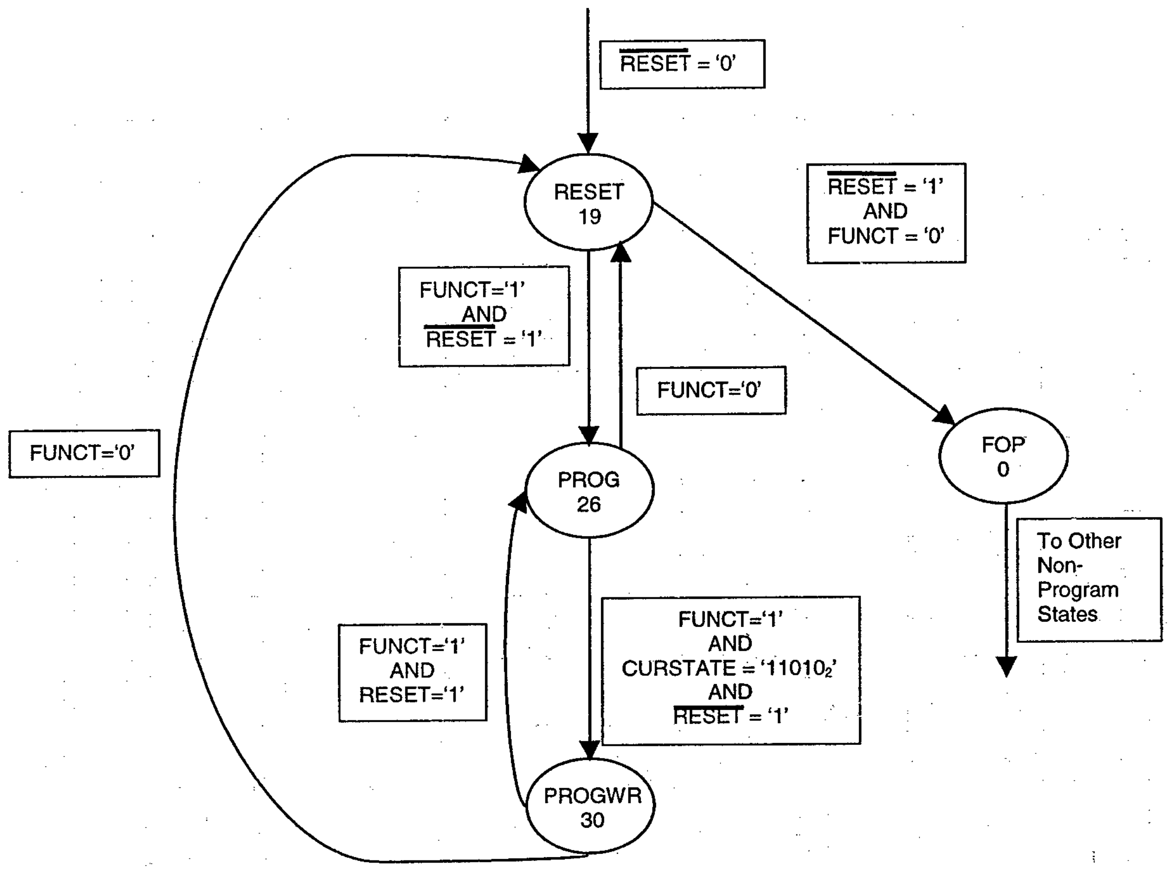
The Temp State element is used to create an additional state for use during the JUMP instruction. Since, there were thirty-two states clarified by the 5-bit CURSTATE signal, I needed to implement another state into the state diagram of the Microdot, but did not want to add another bit to the CURSTATE signal. The Temp State element is a register that latches in the TEMP signal from the Control Logic on every falling-edge of the clock cycle. Only during the JUMP instruction does the TEMP signal become asserted by the Control Logic and thus gets latched through to the Control State Machine, which interprets the TEMP2 signal value and sends the Microdot operations down the correct state diagram path.

The Control State Machine tells the Control Logic, Status Register, ALU Block, and the Status Multiplexor what current state the Microdot is in. In addition, the 5-bit CURSTATE signal is an input/output signal that gets fed back into the Control State

Table 4-6. Control Signals

Operation	LDOP	LDCON1	LDALU	LDCON2	LDTOS	LDMASK	LDTEMP	LDRAM	OFFSET	PCLOAD	PCCOUNT	INCSP	DECSP	CLRSR	TEMP	WAITP	STOREP	SKIPFLAG
Add n	X	X	X		X				X		X							
Subtract n	X	X	X		X				X		X							
Add w/Carry n	X	X	X		X				X		X							
Subtract w/Carry n	X	X	X		X				X		X							
And n	X	X	X		X				X		X							
Or n	X	X	X		X				X		X							
Exclusive-Or n (XOR n)	X	X	X		X				X		X							
Not n	X		X		X						X							
Shift Left (SHL)	X		X		X						X							
Shift Right (SHR)	X		X		X						X							
Load RAM	X							X			X		X					
Pop	X				X						X	X						
Duplicate (DUP)	X				X			X			X		X					
Load	X				X			X			X		X					
Store	X										X						X	
Push c	X	X			X			X			X		X					
SetIO Mask (SETIO)	X	X				X					X							
Wait Mask (WAIT)	X	X									X					X		
Skip Mask (SKIP)	X	X									X							X
Jump Address (JUMP)	X	X		X						X	X				X			
Swap n	X	X			X		X	X	X		X							
Pick n	X	X			X				X		X							
Clear Status Register (CLRSR)	X										X			X				

Machine. The first process after a RESET that occurs with the Microdot is the programming of the on-chip and off-chip SRAM, so that the Microdot can run autonomously from the master microprocessor. The FUNCT and RESET signals play a key role in making sure the programming and transition to running the program execute correctly. Both of these signals are controlled off-chip by the master microprocessor, so at power up the Microdot's elements are just receivers of information. Figure 4-11 shows the state diagram for the programming, which details the states the Control State Machine follows during programming.



$\overline{\text{RESET}}$ = '0'
RESET 19
$\overline{\text{RESET}}$ = '1' AND FUNCT = '0'
FUNCT='1' AND $\overline{\text{RESET}}$ = '1'
FUNCT='0'
FUNCT='0'
FOP 0
PROG 26
To Other Non-Program States
FUNCT='1' AND RESET='1'
FUNCT='1' AND CURSTATE = '11010$_2$' AND $\overline{\text{RESET}}$ = '1'
PROGWR 30

Figure 4-11. State Diagram For Programming

A keynote is that for the Microdot has to have the RESET signal asserted low for a minimum of two clock cycles and then the FUNCT signal can be asserted high. The programming begins once the RESET signal has been de-asserted, while the FUNCT signal remains asserted. This is the only way the start-up and programming of the Microdot has been configured by design. Once the programming is over and the FUNCT signal has been de-asserted, the Microdot will go into the RESET state for a clock cycle and then start to run the instructions that were written into the on-chip and off-chip SRAM.

The Control State Machine element is started up by a 4-bit OPCODE signal value of '0000_2' sent because of the RESET signal being asserted low. The Control State Machine defines the current state of the Microdot and assesses what the next state will be based upon the current state being fed back to itself. Appendix A shows the complete state diagrams that the Microdot uses and is executed by the Control State Machine.

The 4-bit OPCODE signal during the fetch operand (FOP) state tells the Control State Machine what instruction will be executed. After the OPCODE has been delivered, then the Control State Machine will continue down the path of the instruction encoded in the 4-bit operand from the OPCODE Register. The Control State Machine inputs (OPCODE, TEMP2, SKIPFLAG, RESET, and FUNCT) determine the state diagram paths the Microdot follows.

The block that makes the programming of the Microdot possible by accepting an off-chip handshaking signal and returning handshaking signals is the Data Acknowledge Block.

4.8 Data Acknowledge Block

The Data Acknowledge Block's purpose is to produce the handshaking signals used by the master microprocessor, when programming the Microdot. In addition, the block also generates the off-chip SRAM read and write signal from the MEMRW control signal along with the DATAV signal. The DATAV signal comes from the master microprocessor and simply indicates when the data that is sent through the bi-directional ports is valid. When the data is valid, instructions are written to the SRAM cells. After either five nanoseconds or 22 nanoseconds, an acknowledge signal called ACK is sent off-chip back to the master microprocessor. The different delay times depend on whether the memory address location refers to an on-chip SRAM memory address or an off-chip SRAM memory address. Since the Microdot is designed to communicate with a off-chip 15 nanosecond write cycle SRAM, a 22 nanosecond delay is introduced in order for data to be written to and read from the off-chip SRAM without error. The five nanosecond delay is used when writing to on-chip SRAM. The ACK signal lets the master microprocessor know that the Microdot received the data and has successfully written the data into memory. Once the ACK signal is sent to the master microprocessor, the DATAV signal sent from the master microprocessor de-asserts and a delayed time (five nanoseconds or 22 nanoseconds) afterward the ACK signal clears. When the DATAV signal is de-asserted, the master microprocessor sets up the next batch of data to send, while the Microdot increments to the next memory address to be written to. The sending signals back and forth for communication purposes is called handshaking. The only difference between the Acknowledge On-Chip and Acknowledge Off-Chip elements is the time between when the DATAV signal is received and when the ACK signal is sent back. This built in delay from receiving the DATAV signal to sending back the ACK signal is used for giving the SRAM units enough time to write the valid data into memory. Of course, the off-chip SRAM delay needs to longer than for the on-chip SRAM, which takes less time to write to. Figure 4-12 shows all the elements that make up the Data Acknowledge Block.

The Acknowledge Last Address element was designed for the purpose of sending a final prolonged ACK signal back to the master microprocessor when the last memory address has been programmed. This element is used to tell the master microprocessor that it is done programming and all the SRAM cells are full of instructions. When the last memory address is written to, the ACK signal asserts and stays asserted until the FUNCT signal goes low, thus steering the Microdot out of the program state.

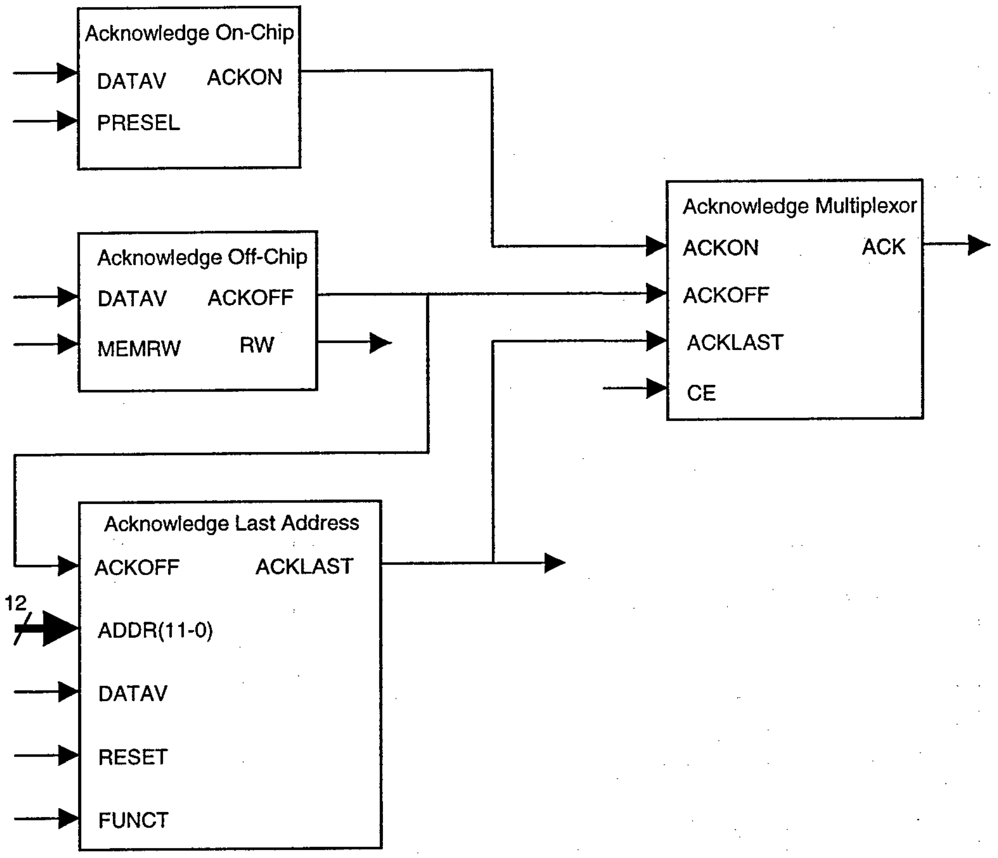


Figure 4-12. Data Acknowledge Block Diagram

The Acknowledge Multiplexor chooses between which delayed acknowledge signal to send through to the ACK output port. A key feature is that the ACKLAST signal is not only one of three input choices for the multiplexor but it also a selector signal. While ACKLAST is asserted high, the output is asserted high. Only when the Acknowledge Last Address element is reset to zero will the ACK signal go back to a low or zero state.

While the Data Acknowledge Block is used during the programming of the Microdot, the Status Multiplexor is designed to give insight to data bus line during the running of the program.

4.9 Status Multiplexor

The Status Multiplexor allows internal signals of the Microdot to be observed. This element is designed for developmental troubleshooting and is not needed for operational use. Figure 4-13 shows the input and output ports of the Status Multiplexor.

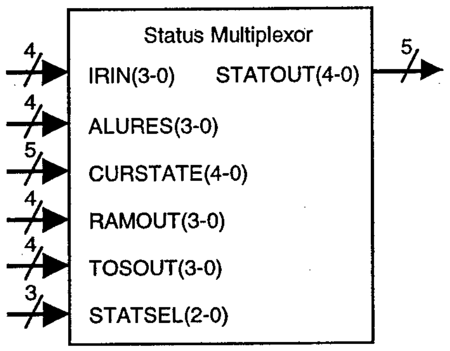


Figure 4-13. Status Multiplexor Diagram

As shown, any of the five main signal lines in the Microdot is observable at any one time. The 3-bit STATSEL signal is manually input through input ports on the Microdot. The Status Multiplexor is controlled from outside the Microdot through use of the selector inputs and can be changed at anytime. The 5-bit output signal STATOUT is sent to output ports, which are visible to the outside world. The STATOUT signal provides visibility to the Instruction Register, ALU Block, Control State Machine, Stack, and the TOS Register.

5. *Testing Procedures and Analysis*

5.1 *VHDL Behavioral and Structural Testing*

The process of testing the Microdot began with the simulation of each component using Very High Speed Integrated Circuit Hardware Description Language (VHDL). The first step in the testing process involved running a complete test of the behavioral VHDL. This step is used to show if the language description of the element accurately portrays the specified requirements. At this point in the testing, no layouts of elements or gates have been chosen, only the behavior of the system is described. Through the VHDL simulator, the functionality of each element was tested. Successful tests result in identifying that the simulator results represent the behavior that I had envisioned for each component. Once each element tests successfully, I began to piece all the components together to make the final working product called the Microdot. In one behavioral VHDL document, I connected the elements together and defined all the inputs, outputs, signals, and port mappings for each element. After all the elements were pieced together, I tested the behavioral VHDL language version of the Microdot using the VHDL simulator. The process of testing the Microdot required running each instruction and running multiple instructions back to back. The instruction and multiple instruction tests verified correct operation of the Microdot behavioral design. Some common errors I encountered included forgetting to use all the inputs specified in the behavioral VHDL file, having the wrong clock edge specified for triggering registers, and not specifying all the possible combinations for a multiplexor component. After learning from these mistakes, I successfully tested all the components and the Microdot at the behavioral

VHDL level. However, there are certain things that cannot be accurately tested at the behavioral VHDL level.

At the behavioral level of abstraction, I tested the programming of the Microdot for the on-chip SRAM. Behavioral VHDL does not involve any timing considerations unless specific time delays are put into the language. I started with no time delays involved, thus the testing of the off-chip SRAM did not seem reasonable. The main reason is that the writing and reading of the off-chip SRAM involves timing issues, since the SRAM is not located on the Microdot. There are two kinds of signals that make timing critical. The master microprocessor sends signals that are interpreted by the Microdot and then sent to the off-chip SRAM. In addition, signals are generated on the Microdot and sent to the off-chip SRAM. Both types of signals need to follow timing specifications for a 15 nanosecond read/write cycle. Therefore, not performing behavioral VHDL testing of the off-chip SRAM without any detailed timing information was a reasonable decision at the behavioral level. The SRAM timing diagrams that were followed through the design of the Microdot for the off-chip SRAM are located in Appendix C. After successfully completing the behavioral VHDL testing of the Microdot, I proceeded to port the behavioral VHDL to structural VHDL.

Structural VHDL uses descriptions of the gates the element needs to make the operations described in the behavioral VHDL become a reality. All the different gates and their connections to each other are documented in a structural VHDL file. A similar test process was used with structural VHDL as with the behavioral VHDL testing; I began with individual component testing of the structural VHDL. Each of the structural VHDL files calls primitive VHDL gate files, which contain timing and functional information. I used timing delays for the testing that were used for gates in the Hewlett-Packard 0.5 micron process. The timing delays were used in a previous thesis that had radiation-hardened standard cells [8]. I used these timing delays in order to ensure success, because the timing delays of the larger radiation-hardened cells was greater than the actual 0.35 micron standard cells I was using. I discovered the difference in timing delays by running HSPICE on each set of standard cells. The extra built-in timing delay was a benefit because all the Microdot signals need to arrive before either a falling or rising clock edge. Since all signals need to arrive on a clock edge, a signal delayed by a half a period or more will cause some problems. Even though my gates are faster based on the HSPICE testing, for testing purposes, using slower times gave me greater confidence in signals arriving on time throughout the Microdot. Once all the components were tested, I began to piece all the components together similar to the behavioral VHDL testing process. The Microdot passed all the instruction and multiple instruction tests on the structural VHDL level. However, there were some testing errors that occurred before successful structural VHDL testing was achieved. Some errors were using the wrong triggered clock edge D flip-flop and improper triggering of test bench inputs. Structural VHDL testing success gave me the conclusion that it was time to start building these elements either by hand layout or through standard cell layout. I hand laid out the Stack, Program Memory, and some Data Acknowledge Block components. Hand layout of the SRAM units was used for the Microdot, since most of the power consumption comes through the SRAM units. Standard cell layout uses up more area, thus has more capacitance and resistance. The higher capacitance and resistance values lead to inevitably using more power to operate an element. Standard cell layout was completed on all the other components for the Microdot using a channel routing software tool known as Octtools. Once I laid out the components using the MAGIC layout software, I extracted a .ext file from MAGIC [25]. Once I converted the .ext file to a .sim file, the testing of the layouts began with IRSIM a switch-level software simulator [26].

5.2 IRSIM Testing

IRSIM is a switch-level simulator that uses resistance and capacitance information from the devices contained in each element. A file is extracted from the layout in MAGIC, and capacitance and resistance values are contained within the file. Therefore, IRSIM is able to calculate relevant timing information based on the device values in the file. However, IRSIM is not the most accurate testing tool, because it does not use logic-cell delays as parameters for its models [26]. IRSIM works on a switch-level basis, which means the signal value can either be a high, low, or unresolved. There are no voltage levels for each signal, only sharp transitions occur which normally would not happen on the Microdot. Each component was tested for functionality and timing by IRSIM. I looked at certain transitions to get a feel for timing scenarios, but I did not base any design decisions on the IRSIM timing data. For the most part, IRSIM testing went quite well with only minor changes needed to correct a few imperfections. One imperfection involved IRSIM trying to model tri-state buffers in the Program Memory and Stack elements. A tri-state buffer has 1.4 volts on the output when it is not enabled. IRSIM does not understand the intermediate voltage levels. Limitations of IRSIM lead me to not test the entire Microdot by excluding the two SRAM elements (Program Memory and Stack). IRSIM can only display 32 signals and has a character limitation on an input line. In other words, the Microdot had too many signals and I could not have run enough sets of inputs. From my experience with IRSIM handling microprocessors, it is very difficult for IRSIM to simulate such a large extracted file with numerous interconnections. However, each of the main blocks that I put together were simulated by IRSIM and passed the functional tests. Therefore, I felt confident that when the Microdot was completely assembled, it would function properly. IRSIM testing results for the Microdot displayed adequate timing information and confidence to proceed to the next more intricate level of software testing called HSPICE.

5.3 HSPICE Testing

The most sophisticated simulation type of testing involved using the HSPICE software tool [27]. HSPICE takes a SPICE file extracted from MAGIC and calculates node voltages, currents, and even power consumption. HSPICE is the most accurate type of software testing available, because it uses model parameters for the N-channel and P-channel transistors. Each N-channel and P-channel transistor is modeled by the parameter file, which is read by HSPICE. I used the level 49 HSPICE model for my tests [27]. I acquired the level 49 model parameters for this particular technology from the fabricator MOSIS [2]. MOSIS is a company that sends educational VLSI design projects to different foundries for fabrication. MOSIS was the direct link for getting the Microdot fabricated. I used the parameters from a wafer run completed in late May of 2000 from Taiwan Semiconductor Manufacturing Corporation (TSMC) using the 0.35 sub-micron process. Each component was tested with HSPICE for functionality and timing information. The most critical timing information, which took several runs of HSPICE, was for the SRAM components. The read and write cycles needed to be timed correctly so false data did not get written into the memory units or read from the memory.

HSPICE has a command to calculate power for each time increment. The power calculation involves consumed power and static power. These calculations are critical to simulation results regarding the Microdot's power consumption profile. The clock frequency that the Microdot is operated at will determine the power consumption of the 4-bit microcontroller. Therefore, HSPICE helped me estimate the true power consumption values the Microdot may require once back from the foundry. Table 5-1 shows the HSPICE results for average power consumption for each of the elements during normal operation.

Table 5-1. Power Consumption by Component

Element Name	Power Consumption During Operation; @ 20 MHz for Clocked Elements	Element Name	Power Consumption During Operation; @ 20 MHz for Clocked Elements
2 to 1 Multiplexor	0.192 mW	OR	0.026 mW
4 to 1 Multiplexor	0.310 mW	Output Buffer	0.034 mW
Acknowledge Last Address	0.068 mW	Output Logic	0.119 mW
Acknowledge Multiplexor	0.046 mW	Output Register	0.418 mW
Acknowledge Off-Chip	0.721 mW	Program Counter Logic	0.309 mW
Acknowledge On-Chip	0.205 mW	Program Counter Multiplexor	0.434 mW
ADDER	0.268 mW	Program Counter State Machine	1.58 mW
ALU Control Unit	0.316 mW	Program Memory	25.1 mW
ALU Overflow	0.169 mW	Shift Left	0.115 mW
ALU Result	0.375 mW	Shift Right	0.116 mW
ALU Zero	0.025 mW	Stack	12.4 mW
ALUCODE Register	0.458 mW	Stack Addressor	0.437 mW
AND	0.020 mW	Stack Logic	0.422 mW
Carry-In Multiplexor	0.023 mW	Stack Multiplexor	0.604 mW
CONSTANT1 Register	0.461 mW	Stack State Machine	0.891 mW
CONSTANT2 Register	0.463 mW	Status Multiplexor	0.247 mW
Control Logic	1.09 mW	Status Register	0.555 mW
Control State Machine	0.642 mW	Subtract Multiplexor	0.259 mW
Event Detection Logic	0.187 mW	Temp State	0.065 mW
Mask Register	0.456 mW	Temporary Register	0.380 mW
Memory Controller	0.011 mW	Top-of-the-Stack Multiplexor	0.415 mW
Memory Multiplexor	0.329 mW	Top-of-the-Stack Register	0.383 mW
NOT	0.011 mW	Two's Complementor	0.059 mW
OPCODE Register	1.02 mW	XOR	0.146 mW

Figure 5-1 shows the average power consumption versus clock frequency achieved through running HSPICE simulations on the Microdot. The figure shows a reduction in power consumption as the clock frequency decreases. The minimum power consumption at 1 kHz is 16.3 milliwatts.

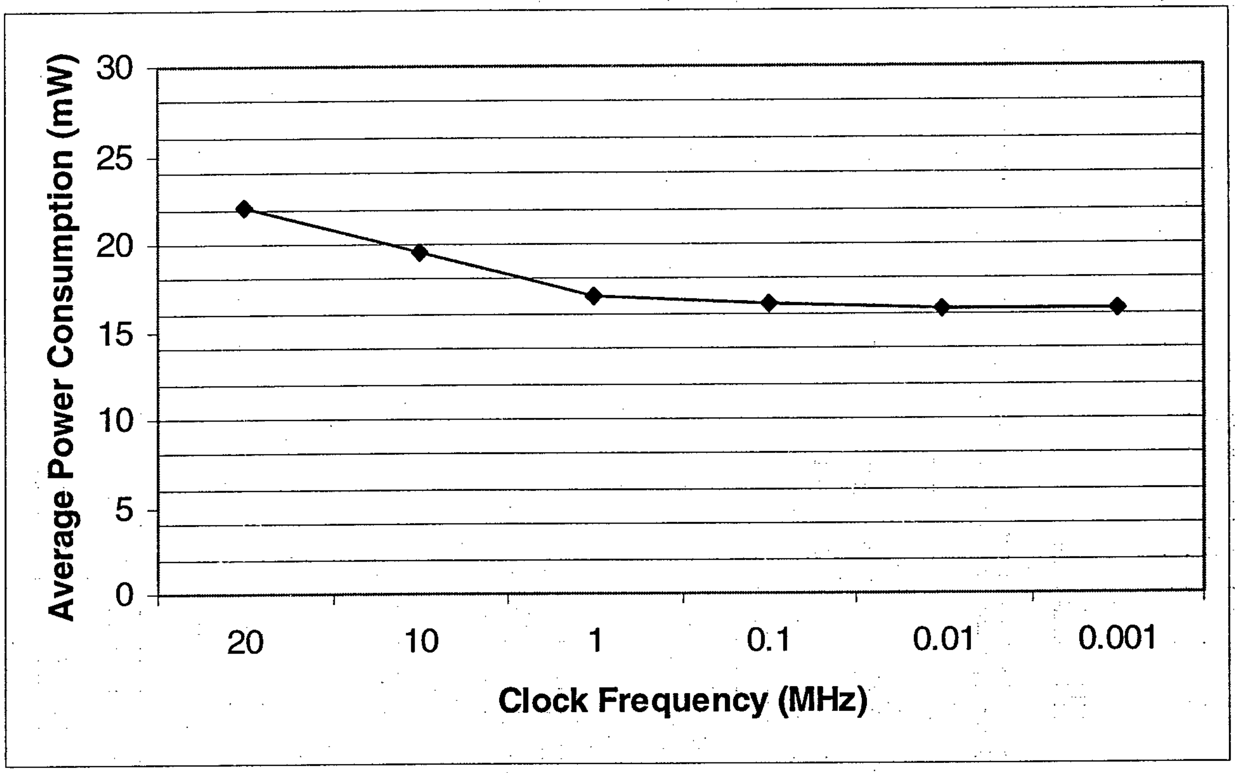


Figure 5-1. Average Power Consumption versus Clock Frequency

Once the Microdot is sent back from the foundry, it is time to test the actual chip on the HP 82000 analyzer. This is the last step in finalizing the validity of the design and the functionality of the Microdot.

5.4 Hewlett Packard 82000 Analyzer Testing

The HP 82000 analyzer is used to test microchips using a DUT board, mainframe, and a UNIX workstation [28]. Software specific to the testing function of the HP 82000 analyzer is accessed through the UNIX workstation. With the HP82000 software test vectors, voltage levels, current levels, pin declarations, power readings, and clock frequency measurements are all accomplished. The HP82000 is used to actually validate the operation and timing of the fabricated Microdot chip. Figure 5-2 shows the test setup of the HP82000 system.

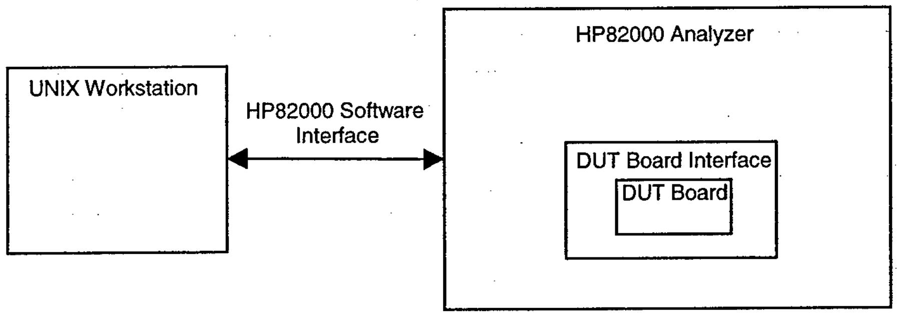


Figure 5-2. HP 82000 Analyzer Testing Setup from [28]

The HP82000 testing began with running a continuity test on the test channels and the DUT board. The continuity test checks the connections between the HP82000 and the pin sockets on the DUT board. The fabricated Microdot chip does not have to be locked in place on the DUT board for the continuity test. The continuity test checks for a short circuit between the tester channels and the DUT board. The continuity test successfully passed on the HP82000, which ensured the DUT board made a good connection with the HP82000. The testing continued with running the Microdot at a slow clock frequency of around hundreds of Hertz, while trying to initialize the Microdot by asserting the RESET signal. Of course, before the instruction set can be tested, the Microdot had to be initialized. Many tests at different frequencies, input voltages, current limits, and input vectors took place in order to get the Microdot to initialize. However, all of my attempts failed to get the Microdot to initialize. I checked the four power and ground pins on the Microdot and found that power and ground were shorted together. After inspection of the MAGIC layout file, I found two elements that had power and ground wired incorrectly. After many attempts to use a laser cutter to try and break the connection of these two elements from the power and ground rail lines, I still was unable to get the Microdot to initialize.

I ran a quiescent current test on the Microdot and found the chip was using 108 milliamps of current in a steady state. The quiescent current measurement went down as I continued to laser cut the two element connections from 108 milliamps to 15 milliamps. As the quiescent current measurement decreased, the resistance measurement between the power and ground rails increased. The resistance measurement started out at 100 ohms and I measured after laser cutting a resistance of 2800 ohms. Even though a sizable resistance was attained between the power and ground rails, a signal transition on the outputs of the Microdot was never measured by the HP82000.

The following process would need to occur in order to fully test the Microdot's operation and functionality. The testing would need to begin by running the following static tests, a continuity test and then a quiescent current test. After successful completion of the continuity test and recording the quiescent current, the active tests may begin. To minimize the complexity of the active tests, beginning with a slow clock speed is preferred, somewhere in the low kilohertz range. After getting initialization to correctly function, the programming of the Microdot would need to be tested. A test involving only programming one instruction into the Microdot and viewing the IRIN, ADDR, and the CURSTATE signal lines is a beginning to testing the Microdot's functionality. After programming successfully, start with programming one instruction into the Microdot and running the instruction. An easy instruction to start with would be the PUSH instruction, since the instruction mainly uses the instruction register and Stack Block. Under the PUSH instruction, the operation of the Stack Block will be tested. After successfully running a Stack instruction, an ALU instruction would be the next test to be executed. During an ALU instruction the output of the ALU Block can be viewed on the STATOUT signal lines. After successful completion of an ALU instruction, testing of the entire instruction set at the same low clock speed should continue. The instruction set can be checked for operational success by the STATOUT signal lines and all the other output lines coming from the Microdot.

The next batch of tests, after successful completion of running the entire instruction set, involves incrementing the clock frequency small amounts to find the maximum clock speed that the Microdot can operate under. Finding the maximum clock speed requires setting the clock frequency to the highest possible frequency until some expected outcome does not occur. Also, these maximum clock speed tests require the clock frequency to be increased until a breaking point occurs. During the process, the Status Multiplexor pins are monitored for different values that should be located on different data lines within the Microdot. Monitoring the STATOUT signal lines is done to better understand the inner workings of the Microdot and to troubleshoot an error if necessary.

Furthermore, minimal power consumption tests need to be run to see what the lowest amount of power consumption the Microdot can function on. A process of lowering the clock speed and running all the Microdot instructions needs to be completed. The goal is to find out the minimum power consumption of the Microdot when operating the clock at a slow clock speed. As the clock speed decreases, the power consumption of the Microdot should decrease.

6. Summary and Conclusions

6.1 Summary

The Microdot is a 4-bit microcontroller that is designed for operations in a space environment. The purpose of the Microdot is to execute programs that deal with data manipulation from sensors on-board a satellite. The main role is to process and temporarily store the data and report if any of the data is out of limits set by the programmer. The execution of a reporting program is only one of the many scenarios that the Microdot could be used for. This thesis used documentation from the Air Force Research Laboratory located at Kirtland AFB NM as a starting point to begin the Microdot design [1, 11]. The work that was completed for this thesis is a building block to fabricating a prototype of the Microdot. New concepts and design decisions were made throughout the design of the Microdot. The LDRAM instruction was added along with ALU Block control signals to minimize power consumption of the Microdot. Finally, all the Microdot elements were integrated to achieve the prototype product and HSPICE simulations were used to validate the operation and specifications of the Microdot. The fabricated Microdot was unable to be tested for functionality due to two elements having the power and ground rail connections switched.

6.2 Conclusions

The results that were displayed in Chapter Five demonstrate the feasibility of a small and efficient 4-bit microcontroller that can meet mission requirements for space applications. The lowest power consumption of the Microdot during operation using HSPICE simulation runs was 15 microwatts. Also, the highest operating frequency for the Microdot using HSPICE simulation runs was 20 MHz. The lowest tested operating frequency for the Microdot was 1 kHz. In addition, the Microdot was simulated using HSPICE and all of the 23 instructions were performed correctly and without error. Therefore, I believe that due to these results further research should be continued in this area.

6.3 Lessons Learned

From the beginning to the end of the design process, several lessons were learned. One crucial lesson is to learn the software tools thoroughly, because most thesis designs are more complex than class designs. Complicated designs can make the automation of going from behavioral to structural VHDL code more difficult and, thus, take more time to achieve.

Another lesson learned is attaining the knowledge of the Lager Octtools software and all the files that the software needs to work properly. When using a different standard cell library than the traditional Lager library, it is critical to know how to set up each standard cell MAGIC layout. There are many particular dimensions that need to be followed for the labels, power and ground rails, and overall cell size. Appendix D contains a tutorial for using the Lager Octtools software with new standard cell libraries.

A better routing tool to be used with standard cell libraries would greatly reduce the time and area of designs in the AFIT VLSI lab. The Lager Octtools software currently routes between standard cells using channel routing. The software creates extra channels where only routed lines run. Area is wasted by this technique, since wire connections could be accomplished by going over cells in higher levels of metal.

When trying to meet a fabrication run date, set aside at least a couple of weeks to test the design. I did test the Microdot before sending it to fabrication and everything worked properly. However, I discovered when the chip returned from the foundry, there were interconnections and a state transition that my simulation tests had not covered. There are unlimited amounts of tests to run on any chip design, so choosing which tests to run is critical. I chose to test numerous functions yielding a high percent fault coverage, however without a large amount of time it is sometimes difficult to reach a 100 percent or complete fault coverage for an ASIC.

6.4 Recommendations for Future Research

I recommend that the Microdot design be re-fabricated in the 0.35 sub-micron TSMC process using commercial standard cells. I have instituted the following corrections to the MAGIC layout file of the Microdot, so correct operation will occur. I have corrected the two element connections to power and ground, therefore correcting the short between power and ground. I also corrected a state transition from PROG (state 26) to PROGWR (state 30) in the Control State Machine element, which hindered the programming of the Microdot. HSPICE simulation runs of all the instructions at varying frequencies have validated the corrected Microdot design. The fabrication of the corrected Microdot design will give a baseline to work from in the future. The main elements that consumed power in the HSPICE simulation runs were the SRAM units (Program Memory and Stack). A redesign of the SRAM cells and research in this area would help achieve the goal of the Microdot, which is to consume low amounts of power. In addition, a radiation tolerant Microdot design using radiation hardened standard cells should be created in the 0.5 sub-micron Hewlett-Packard (HP) process. The 0.35 sub-micron TSMC process does not guarantee the threshold voltage levels of annular transistors. The HP process has the ability to fabricate the annular transistor without any limitations and therefore may be the best chance at making an acceptable radiation tolerant design for the Microdot. A comparison of the two different Microdots from the HP and TSMC process would have great merit. Specifications such as radiation tolerance, power consumption, and area could be compared and optimized for each design. A fabrication process and design could be decided on by continuing the research of the Microdot in this direction.

Appendix A. Microdot Layout and State Diagrams

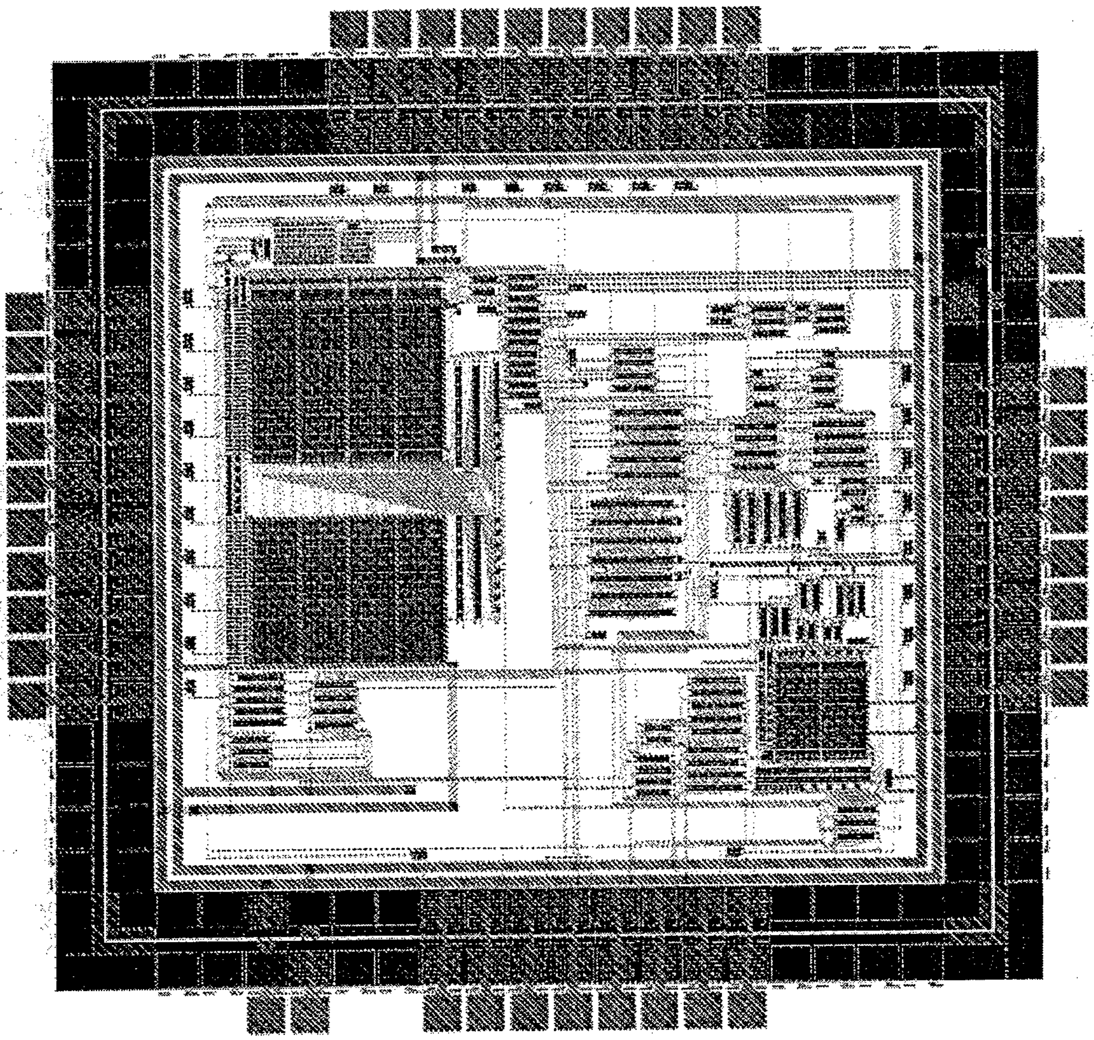
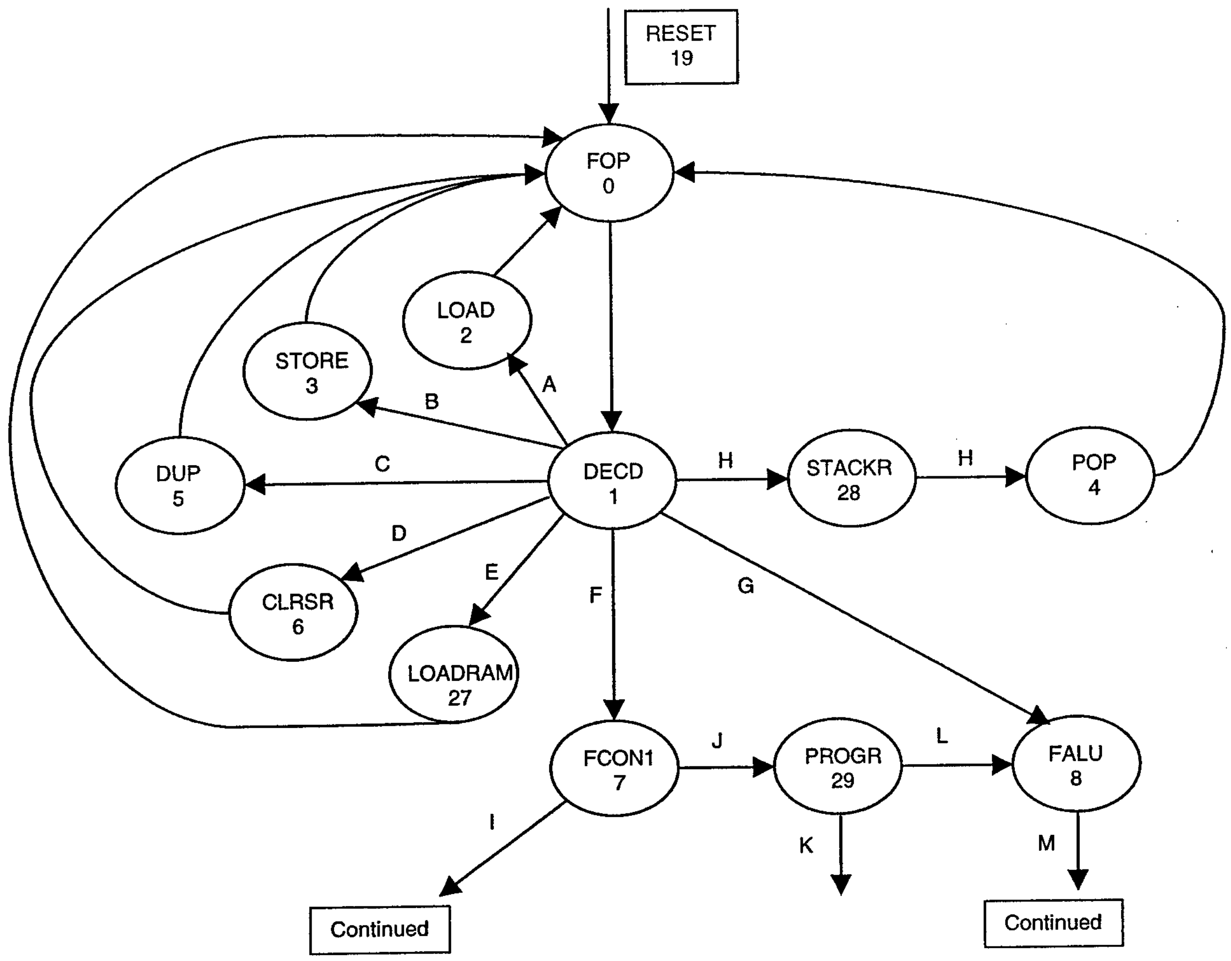
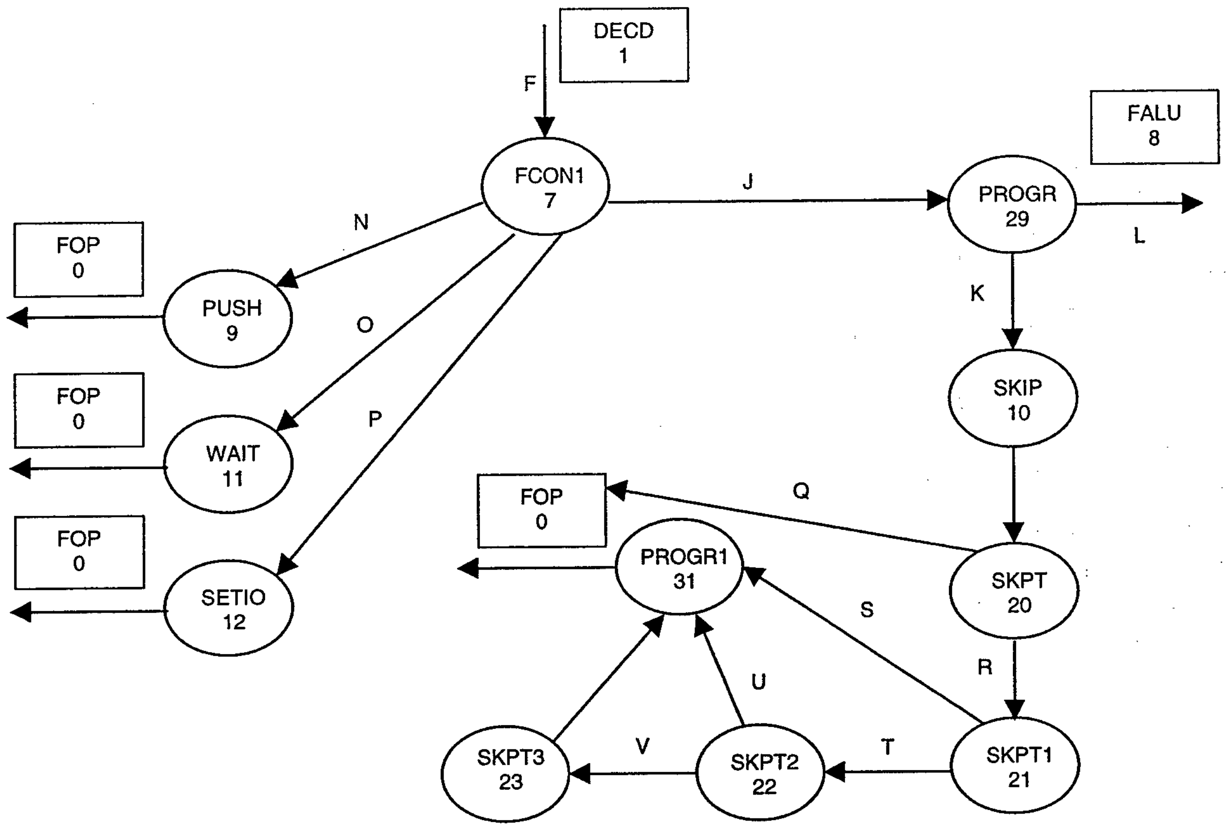


Figure A-1. Microdot Layout



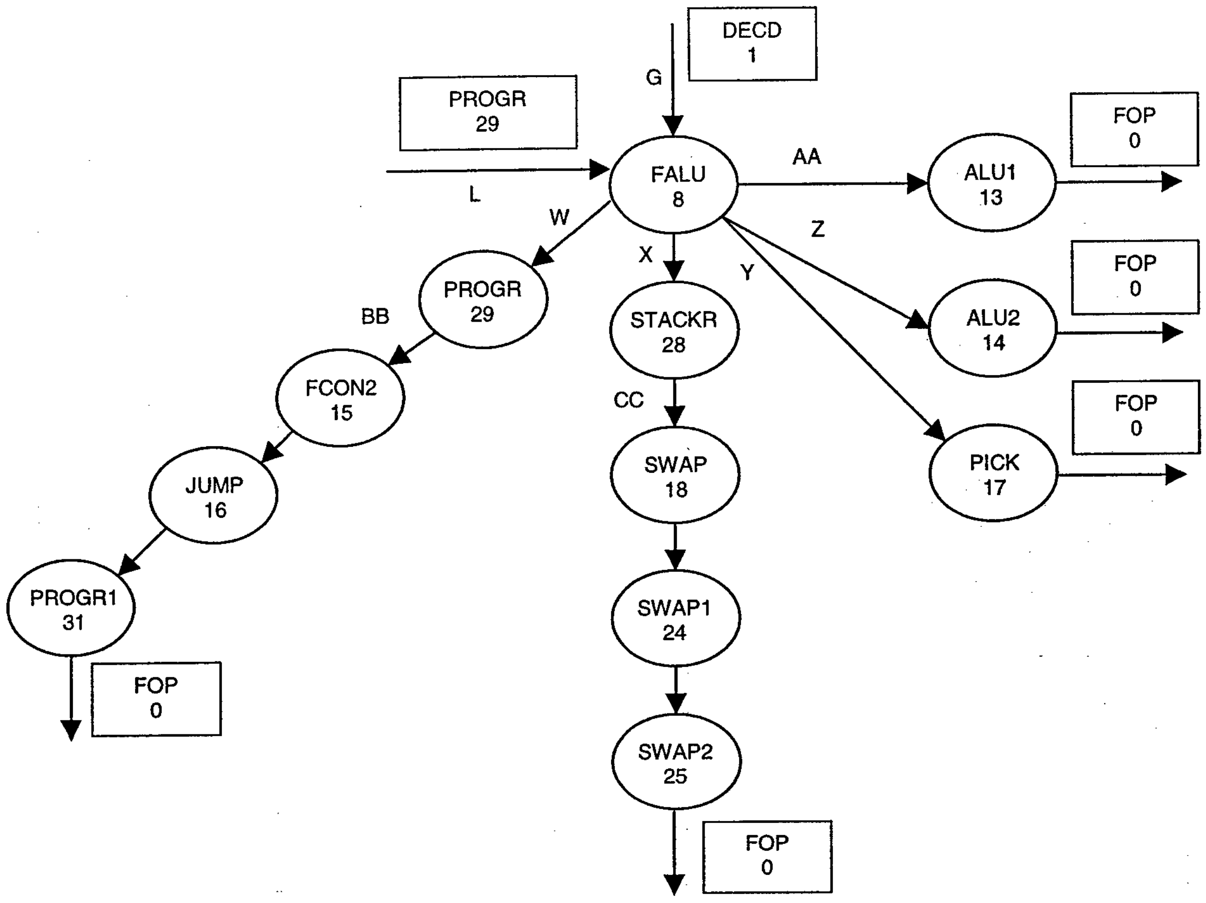
Path Name	OPCODE = (Binary Code)	Path Name	OPCODE = (Binary Code)
A	‘0110’	H	‘0100’
B	‘0111’	I	‘1000’ or ‘1001’ or ‘1010’
C	‘0101’	J	‘0000’ or ‘1011’ or ‘1100’ or ‘1101’ or ‘1110’
D	‘1111’	K	‘1011’
E	‘0010’	L	‘0000’ or (‘1100’ and TEMP2 = ‘1’) or ‘1101’ or ‘1110’
F	‘0000’ or ‘1000’ or ‘1001’ or ‘1010’ or ‘1011’ or ‘1100’ or ‘1101’ or ‘1110’	M	‘0000’ or ‘0001’ or (‘1100’ and TEMP2 = ‘1’) or ‘1101’ or ‘1110’
G	‘0001’		

Figure A-2. Microdot State Diagram (1 of 3)



Path Name	OPCODE = (Binary Code)	Path Name	OPCODE = (Binary Code)
F	'0000' or '1000' or '1001' or '1010' or '1011' or '1100' or '1101' or '1110'	Q	SKIPFLAG = '0'
J	'0000' or '1011' or '1100' or '1101' or '1110'	R	SKIPFLAG = '1'
K	'1011'	S	'0100' or '0101' or '0111' or '1111'
L	'0000' or ('1100' and TEMP2 = '1') or '1101' or '1110'	T	'0001' or '1000' or '1001' or '1010' or '1011' or '1100' or '1101' or '1110'
N	'1000'	U	'0001' or '1000' or '1001' or '1010'
O	'1010'	V	'1011' or '1100' or '1101' or '1110'
P	'1001'		

Figure A-3. Microdot State Diagram (2 of 3)



Path Name	OPCODE = (Binary Code)	Path Name	OPCODE = (Binary Code)
G	'0001'	Z	'0000'
L	'0000' or ('1100' and TEMP2 = '1') or '1101' or '1110'	AA	'0001'
W	'1100'	BB	'1100' and TEMP2 ='0'
X	'1101'	CC	'1101'
Y	'1110'		

Figure A-4. Microdot State Diagram (3 of 3)

Appendix B. Microdot Signal Table and Specifications

Table B-1. Microdot Signal Table

Signal Name	# of Bits	FROM	TO	FUNCTION
ACK	1	Acknowledge Multiplexor	OFF-CHIP	Handshaking signal used during programming
ACKLAST	1	Acknowledge Last Address	Acknowledge Multiplexor; Control Logic	Turns off programming after reaching the 4096th memory location
ACKOFF	1	Acknowledge Off-Chip	Acknowledge Multiplexor; Acknowledge Last Address	Delayed handshaking signal used when programming the off-chip SRAM unit
ACKON	1	Acknowledge On-Chip	Acknowledge Multiplexor	Delayed handshaking signal used when programming the on-chip SRAM unit
ADDCSEL	1	ALU Control Unit	Carry-In Multiplexor; Adder	Signals that the Add with Carry instruction is being performed
ADDOUT	4	Adder	ALU Result	Result of the Adder element
ADDR	12	Program Counter State Machine	Program Counter Multiplexor; Program Counter Logic; Program Memory; Memory Controller; Acknowledge Last Address; OFF-CHIP	Address which decides which SRAM cell to write to or read from
ADDRIN	12	Program Counter Logic	Program Counter Multiplexor	ADDR incremented by one; provides next address
ADDRMX	12	Program Counter Multiplexor	Program Counter State Machine	Selected address to get sent to SRAM units
ADDSEL	1	ALU Control Unit	Carry-In Multiplexor; Adder	Signals that the Add instruction is being performed
ADDSUB	1	ALU Control Unit	Subtract Multiplexor; ALU Overflow; Two's Complementor	Declares whether the ALU operation is addition or subtraction
ALUCODE	4	ALUCODE Register	Program Counter Multiplexor; ALU Control Unit; ALU Result; Control Logic; Status Register;	Declares which ALU operation will occur. Used in the JUMP instruction sequence.
ALURES	4	ALU Result	ALU Zero; 2 to 1 Multiplexor; Status Multiplexor	The selected output of the ALU Block
ANDOUT	4	AND	ALU Result	Result of the AND operation
ANDSEL	1	ALU Control Unit	AND	Turns the AND element on and off
CARRY	1	ALU Control Unit	Status Register	Gives resulting carry bit
CBIT	1	Status Register	Control Logic; Carry-In Multiplexor; ALU Control Unit; Shift Left; Shift Right	Current carry bit
CE	1	Memory Controller	Program Memory; Memory Multiplexor; Acknowledge Multiplexor	Enables either the on-chip or off-chip SRAM

Signal Name	# of Bits	FROM	TO	FUNCTION
CLK	1	OFF-CHIP	Control State Machine; Status Register; Temp State; Mask Register; Output Register; Program Memory; Program Counter State Machine; OPCODE Register; ALUCODE Register; CONSTANT1 Register; CONSTANT2 Register; Temporary Register; Top-of-the-Stack Register; Stack; Stack State Machine	System Clock
CLRSR	1	Control Logic	Status Register	Resets the Status Register to all zeros
CONST1	4	CONSTANT1 Register	Program Counter Multiplexor; 4 to 1 Multiplexor; Control Logic; Mask Register; Event Detection Logic	All-purpose instruction register output
CONST2	4	CONSTANT2 Register	Program Counter Multiplexor	Used for JUMP Instruction
COUT	1	ADDER	ALU Control Unit	Carry bit result from addition operation
CURSTATE	5	Control State Machine	Control Logic; Status Multiplexor; Status Register; ALU Control Unit	Current state bits- keeps track of which state the Microdot is in
DATASEL1	1	Control Logic	4 to 1 Multiplexor; Top-of-the-Stack Multiplexor	Select bit for both multiplexors
DATASEL2	1	Control Logic	2 to 1 Multiplexor	Select bit for the multiplexor
DATAV	1	OFF-CHIP	Acknowledge On-Chip; Acknowledge Off-Chip; Acknowledge Last Address; Program Memory	Handshaking signal used during programming
DECSP	1	Control Logic	Stack Multiplexor; Stack Logic	Tells Stack Block to decrement current stack address
DOFFOUT	4	OFF-CHIP	Memory Multiplexor	Output bits from Off-chip SRAM
DONOUT	4	Program Memory	Memory Multiplexor	Output bits from On-chip SRAM
EVENT	1	Event Detection Logic	Control State Machine	Used during WAIT instruction; signals Microdot to stop waiting
FUNCT	1	OFF-CHIP	Control State Machine; Acknowledge Last Address	Triggers the beginning and end of programming
INCSP	1	Control Logic	Stack Multiplexor; Stack Logic	Tells Stack Block to increment current stack address
INPUT	4	OFF-CHIP	Program Memory; Memory Multiplexor; 4 to 1 Multiplexor; Event Detection Logic;	4-bit input/output data path

Signal Name	# of Bits	FROM	TO	FUNCTION
IRIN	4	Memory Multiplexor	OPCODE Register; ALUCODE Register; CONSTANT1 Register; CONSTANT2 Register; Status Multiplexor	Instruction register data path
LDALU	1	Control Logic	ALUCODE Register; Program Memory	Signals loading of the ALUCODE Register
LDCONST1	1	Control Logic	CONSTANT1 Register	Signals loading of the CONSTANT1 Register
LDCONST2	1	Control Logic	CONSTANT2 Register	Signals loading of the CONSTANT2 Register
LDMASK	1	Control Logic	Mask Register	Signals when to load register
LDOP	1	Control Logic	OPCODE Register	Signals when to load register
LDRAM	1	Control Logic	Stack	Signals when to write to stack
LDTEMP	1	Control Logic	Temporary Register	Signals when to load register
LDTOS	1	Control Logic	Top-of-the-Stack Register	Signals when to load register
MASKOUT	4	Mask Register	Output Logic	Sets which bits are input or output bits
MEMRW	1	Control Logic	Program Memory; Acknowledge Off-Chip	Signals either to read or write from memory
MUX2OUT	4	2 to 1 Multiplexor	Top-of-the-Stack Multiplexor	output of multiplexor
MUX4OUT	4	4 to 1 Multiplexor	Top-of-the-Stack Multiplexor	output of multiplexor
NBIT	1	Status Register	Control Logic	Signals whether result from ALU is negative
NEGRES	1	ALU Zero	Status Register	Signals whether result from ALU is negative
NOTOUT	4	NOT	ALU Result	Result of the NOT operation
NOTSEL	1	ALU Control Unit	NOT	Turns the NOT element on and off
OFFSET	7	Control Logic	Stack Addressor	One of two 7-bit operands for 7-bit adder; gives ability to reach any of the 128 stack addresses within the stack
OPCODE	4	OPCODE Register	Control State Machine; ALU Control Unit; ALU Result; Control Logic	Gives the operational code, which determines which instruction to run
OROUT	4	OR	ALU Result	Result of the OR operation
ORSEL	1	ALU Control Unit	OR	Turns the OR element on and off
OUT	4	Output Logic	Output Register	Sends data off-chip
OUTPUT	4	Output Register	OFF-CHIP	Sends selected output bits off-chip
OVERRIDE	1	OFF-CHIP	Memory Multiplexor	By-passes both SRAM units; gives ability to load instructions manually
OVFLW	1	ALU Overflow	Status Register	Signals whether result from ALU is an overflow condition
PCCOUNT	1	Control Logic	Program Counter Multiplexor	Tells multiplexor to send through incremented address value

Signal Name	# of Bits	FROM	TO	FUNCTION
PCLOAD	1	Control Logic	Program Counter Multiplexor	Tells multiplexor to load 12-bit address from the instruction registers- used during JUMP instruction
PCSET	1	Control Logic	Program Counter State Machine	Resets address to "000000000000" after programming is complete
PRESEL	1	Program Memory	Acknowledge On-Chip	Tells handshake signal to go high when Program Memory is ready to write to SRAM cell
PRESELCNTL	1	Control Logic	Stack	Tells stack when to read from SRAM cell
RAMNOUT	4	Subtract Multiplexor	ADDER	4-bit operand to ADDER
RAMOUT	4	Stack	Temporary Register; Subtract Multiplexor; ALU Overflow; Two's Complementor; OR; AND; XOR; Status Multiplexor	Output from Stack
RAMQ	4	Two's Complementor	Subtract Multiplexor; ALU Overflow	Complemented version of RAMOUT data
RESET	1	OFF-CHIP	Stack State Machine; Mask Register; OPCODE Register; Program Counter State Machine; Control State Machine; Acknowledge Last Address	Initializing signal
RW	1	Acknowledge Off-Chip	OFF-CHIP	Read/Write signal for off-chip SRAM
SHLC	1	Shift Left	ALU Control Unit	Carry result from shift operation
SHLOUT	4	Shift Left	ALU Result	Result of shift left operation
SHLSEL	1	ALU Control Unit	Shift Left	Turns the Shift Left element on and off
SHRC	1	Shift Right	ALU Control Unit	Carry result from shift operation
SHROUT	4	Shift Right	ALU Result	Result of shift right operation
SHRSEL	1	ALU Control Unit	Shift Right	Turns the Shift Right element on and off
SKIPFLAG	1	Control Logic	Control State Machine	Used during SKIP instruction as a go or no-go flag
SPIN	7	Stack State Machine	Stack Logic; Stack Addressor; Stack Multiplexor	One of two 7-bit operands for 7-bit adder; gives ability to reach any of the 128 stack addresses within the stack
SPMX	7	Stack Multiplexor	Stack State Machine	input to stack state machine
SPOUT	7	Stack Logic	Stack Multiplexor	Incremented by one or decremented by one SPIN address
STAKADR	7	Stack Addressor	Stack	Result of adding OFFSET and SPIN- current stack address
STATOUT	5	Status Multiplexor	OFF-CHIP	Used to view different internal data buses within the Microdot

Signal Name	# of Bits	FROM	TO	FUNCTION
STATSEL	3	OFF-CHIP	Status Multiplexor	Select lines to control which data buses to output on STATOUT
STOREP	1	Control Logic	Output Register	Used to send data off-chip, during the STORE instruction
SUBCSEL	1	ALU Control Unit	Carry-In Multiplexor; Adder	Signals that the Subtract with Carry instruction is being performed
SUBSEL	1	ALU Control Unit	Carry-In Multiplexor; Adder	Signals that the Subtract instruction is being performed
TEMP	1	Control Logic	Temp State	temporary control signal to create an extra state during the JUMP instruction
TEMP2	1	Temp State	Control State Machine	tells state machine to go to state 15-FCON2; during the JUMP instruction only
TEMPOUT	4	Temporary Register	2 to 1 Multiplexor	Used to store RAMOUT during SWAP instruction
TOSIN	4	Top-of-the-Stack Multiplexor	Top-of-the-Stack Register	Used to transfer data into the TOS Register (Cache)
TOSOUT	4	Top-of-the-Stack Register	Stack; Status Multiplexor; ADDER; OR; NOT; XOR; AND; XOR; Shift Left; Shift Right; Event Detection Logic; Output Logic	Used to write data to stack and perform ALU operations on data; also used for miscellaneous operations
VBIT	1	Status Register	Control Logic	Tells control logic whether last ALU operation had an overflow condition
WAITP	1	Control Logic	Event Detection Logic	Signals wait for an event on the selected INPUT bits
XOROUT	4	XOR	ALU Result	Result from the XOR operation
XORSEL	1	ALU Control Unit	XOR	Turns the XOR element on and off
ZBIT	1	Status Register	Control Logic	Tells control logic whether last ALU operation had a zero result
ZERORES	1	ALU Zero	Status Register	Signals a zero result from the ALU Result

Table B-2. Microdot Specifications

Package Type	DIP40
Supply Voltage	3.3VDC
Average Power (Simulated):	16.3 mW
Average Set-Up Power (Simulated):	115 mW
Core Transistor Count:	41,872
Total Chip Area:	2243 x 2152 microns = 4.83 mm^2
Pin Count: (40 Pins Total)	4 Vdd and GND 1 CLK (Input) 1 DATAV (Input) 4 DOFFOT (Inputs) 1 FUNCT (Input) 1 OVERRIDE (Input) 1 RESET (Input) 3 STATSEL (Inputs) 4 INPUT (Inputs/Outputs) 1 ACK (Output) 12 Address (Outputs) 1 CE (Output) 1 RW (Output) 5 STATOUT (Outputs)

Appendix C. SRAM Specifications and Timing Diagrams

CYPRESS

CY7C168A

4Kx4 RAM

Features

- Automatic power-down when deselected
- CMOS for optimum speed/power
- High speed
  - t_{AA} = 15 ns
- Low active power
  - 633 mW
- Low standby power
  - 110 mW
- TTL-compatible inputs and outputs
- V_{IH} of 2.2V
- Capable of withstanding greater than 2001V electrostatic discharge

Functional Description

The CY7C168A is a high-performance CMOS static RAM organized as 4096 by 4 bits. Easy memory expansion is provided by an active LOW Chip Enable ($\overline{CE}$) and three-state drivers. The CY7C168A has an automatic power-down feature, reducing the power consumption by 77% when deselected.

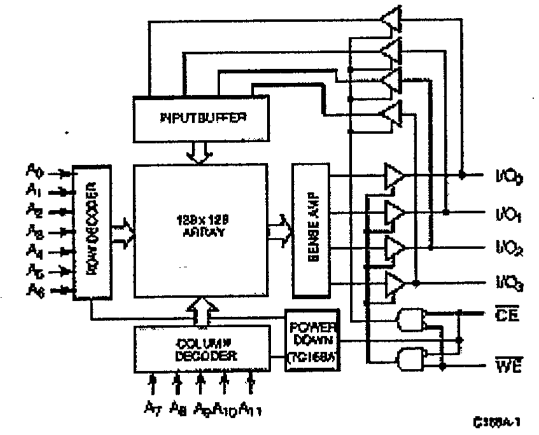
Writing to the device is accomplished when the Chip Select ($\overline{CE}$) and Write Enable ($\overline{WE}$) inputs are both LOW. Data on the four data input/output pins (I/O$_0$ through I/O$_3$) is written into the memory location specified on the address pins (A_0 through A_{11}).

Reading the device is accomplished by taking the Chip Enable ($\overline{CE}$) LOW, while Write Enable ($\overline{WE}$) remains HIGH. Under these conditions, the contents of the location specified on the address pins will appear on the four data input/output pins (I/O$_0$ through I/O$_3$).

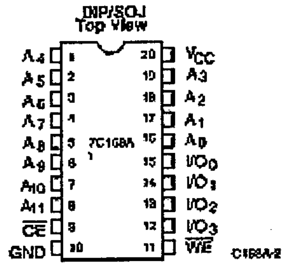
The input/output pins remain in a high-impedance state when Chip Enable ($\overline{CE}$) is HIGH or Write Enable ($\overline{WE}$) is LOW.

A die coat is used to insure alpha immunity.

Logic Block Diagram



Pin Configurations



Selection Guide

		7C168A-15	7C168A-20	7C168A-25	7C168A-35	7C168A-45
Maximum Access Time (ns)		15	20	25	35	45
Maximum Operating Current (mA)	Commercial	115	90	90	90	80
	Military	-	100	100	100	100

Maximum Ratings

(Above which the useful life may be impaired. For user guidelines, not tested.)

Storage Temperature	−65°C to +150°C
Ambient Temperature with Power Applied	−55°C to +125°C
Supply Voltage to Ground Potential (Pin 20 to Pin 10)	−0.5V to +7.0V
DC Voltage Applied to Outputs in High Z State	−0.5V to +7.0V
DC Input Voltage	−3.0V to +7.0V
Output Current into Outputs (Low)	20 mA
Static Discharge Voltage (per MIL-STD-883, Method 3015)	>2001V
Latch-Up Current	>200 mA

Operating Range

Range	Ambient Temperature	V_{CC}
Commercial	0°C to +70°C	5V ± 10%
Military[1]	−55°C to +125°C	5V ± 10%

Electrical Characteristics Over the Operating Range[2]

Parameter	Description	Test Conditions		7C168A-15 Min.	7C168A-15 Max.	7C168A-20 Min.	7C168A-20 Max.	Unit
V_{OH}	Output HIGH Voltage	V_{CC} = Min., I_{OH} = −4.0 mA		2.4		2.4		V
V_{OL}	Output LOW Voltage	V_{CC} = Min., I_{OL} = 8.0 mA			0.4		0.4	V
V_{IH}	Input HIGH Voltage			2.2	V_{CC}	2.2	V_{CC}	V
V_{IL}	Input LOW Voltage[3]			−0.5	0.8	−0.5	0.8	V
I_{IX}	Input Load Current	GND ≤ V_I ≤ V_{CC}		−10	+10	−10	+10	µA
I_{OZ}	Output Leakage Current	GND ≤ V_O ≤ V_{CC}, Output Disabled		−10	+10	−10	+10	µA
I_{OS}	Output Short Circuit Current[4]	V_{CC} = Max., V_{OUT} = GND			−350		−350	mA
I_{CC}	V_{CC} Operating Supply Current	V_{CC} = Max., I_{OUT} = 0 mA	Com'l		115		90	mA
			Mil		-		100	
I_{SB1}	Automatic $\overline{CE}$ Power-Down Current	Max. V_{CC}, $\overline{CE}$ ≥ V_{IH}	Com'l		40		40	mA
			Mil		-		40	
I_{SB2}	Automatic $\overline{CE}$ Power-Down Current	Max. V_{CC}, $\overline{CE}$ ≥ V_{CC} − 0.3V	Com'l		20		20	mA
			Mil		-		20	

Notes:
1. T_A is the "instant on" case temperature.
2. See the last page of this specification for Group A subgroup testing information.
3. V_{IL} min. = −3.0V for pulse durations less than 30 ns.
4. Not more than 1 output should be shorted at one time. Duration of the short circuit should not exceed 30 seconds

Electrical Characteristics Over the Operating Range[2] (continued)

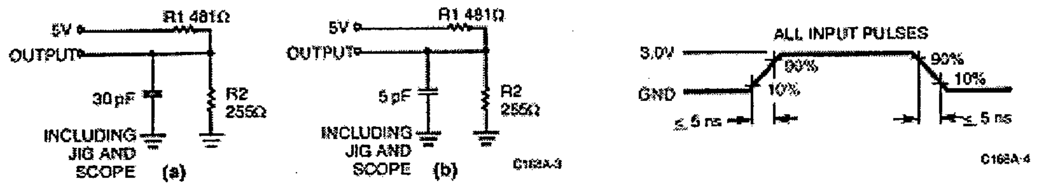
Parameter	Description	Test Conditions		7C168A-25 Min.	7C168A-25 Max.	7C168A-35 Min.	7C168A-35 Max.	7C168A-45 Min.	7C168A-45 Max.	Unit
V_{OH}	Output HIGH Voltage	V_{CC} = Min., I_{OH} = −4.0 mA		2.4		2.4		2.4		V
V_{OL}	Output LOW Voltage	V_{CC} = Min., I_{OL} = 8.0 mA			0.4		0.4		0.4	V
V_{IH}	Input HIGH Voltage			2.2	V_{CC}	2.2	V_{CC}	2.2	V_{CC}	V
V_{IL}	Input LOW Voltage[3]			−0.5	0.8	−0.5	0.8	−0.5	0.8	V
I_{IX}	Input Load Current	GND ≤ V_I ≤ V_{CC}		−10	+10	−10	10	−10	10	μA
I_{OZ}	Output Leakage Current	GND ≤ V_O ≤ V_{CC} Output Disabled		−10	+10	−50	50	−50	50	μA
I_{OS}	Output Short Circuit Current[4]	V_{CC} = Max., V_{OUT} = GND			−350		−350		−350	mA
I_{CC}	V_{CC} Operating Supply Current	V_{CC} = Max., I_{OUT} = 0 mA	Com'l		90		90		90	mA
			Mil		100		100		100	
I_{SB1}	Automatic $\overline{CE}$ Power-Down Current	Max. V_{CC}, $\overline{CE}$ ≥ V_{IH}	Com'l		20		20		20	mA
			Mil		20		20		20	
I_{SB2}	Automatic $\overline{CE}$ Power-Down Current	Max. V_{CC}, $\overline{CE}$ ≥ VCC − 0.3 V	Com'l		20		20		20	mA
			Mil		20		20		20	

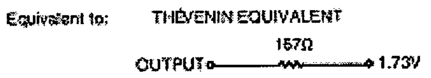
Capacitance[5]

Parameter	Description	Test Conditions	Max.	Unit
C_{IN}	Input Capacitance	T_A = 25°C, f = 1 MHz, V_{CC} = 5.0V	10	pF
C_{OUT}	Output Capacitance		10	pF

Note:
5. Tested initially and after any design or process changes that may affect these parameters.

AC Test Loads and Waveforms



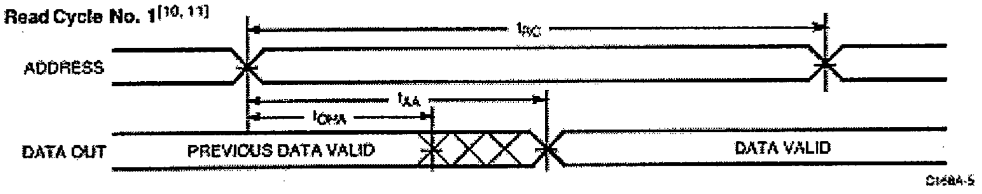


Switching Characteristics Over the Operating Range[2,5]

Parameter	Description	7C168A-15		7C168A-20		7C168A-25		7C168A-35		7C168A-45		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
READ CYCLE												
t_{RC}	Read Cycle Time	15		20		25		35		45		ns
t_{AA}	Address to Data Valid		15		20		25		35		45	ns
t_{OHA}	Output Hold from Address Change	5		5		5		5		5		ns
t_{ACE}	Power Supply Current		15		20		25		35		45	ns
t_{LZCE}	$\overline{CE}$ LOW to Low Z[7]	5		5		5		5		5		ns
t_{HZCE}	$\overline{CE}$ HIGH to High Z[7, 8]		8		8		10		15		15	ns
t_{PU}	$\overline{CE}$ LOW to Power Up	0		0		0		0		0		ns
t_{PD}	$\overline{CE}$ HIGH to Power-Down		15		20		20		20		25	ns
t_{RCS}	Read Command Set-Up	0		0		0		0		0		ns
t_{RCH}	Read Command Hold	0		0		0		0		0		ns
WRITE CYCLE[9]												
t_{WC}	Write Cycle Time	15		20		20		25		40		ns
t_{SCE}	$\overline{CE}$ LOW to Write End	12		15		20		25		30		ns
t_{AW}	Address Set-Up to Write End	12		15		20		25		30		ns
t_{HA}	Address Hold from Write End	0		0		0		0		0		ns
t_{SA}	Address Set-Up to Write Start	0		0		0		0		0		ns
t_{PWE}	$\overline{WE}$ Pulse Width	12		15		15		20		20		ns
t_{SD}	Data Set-Up to Write End	10		10		10		15		15		ns
t_{HD}	Data Hold from Write End	0		0		0		0		0		ns
t_{LZWE}	$\overline{WE}$ HIGH to Low Z[7]	7		7		7		5		5		ns
t_{HZWE}	$\overline{WE}$ LOW to High Z[7, 8]	5		5		5		5		10		ns

Switching Waveforms

Read Cycle No. 1[10, 11]

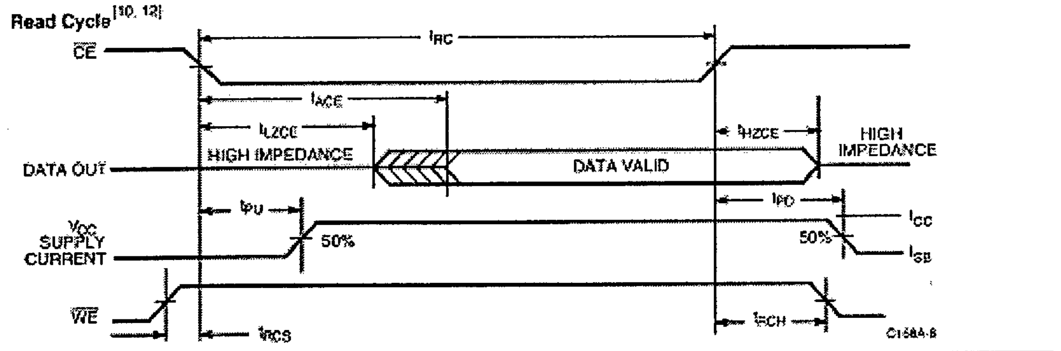


Notes:

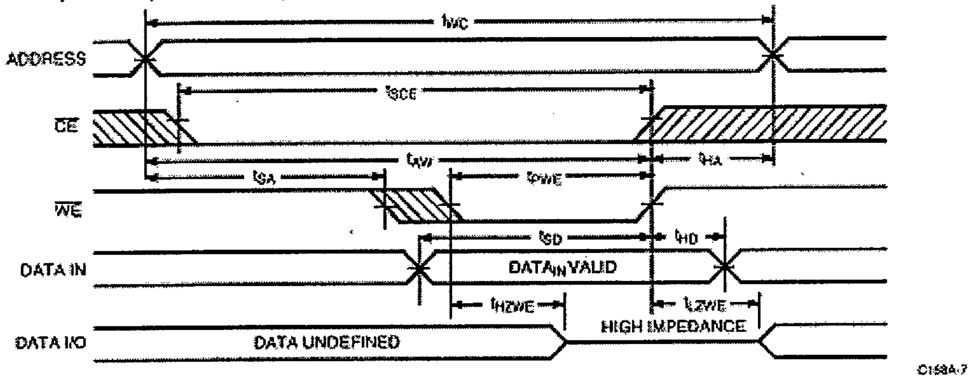
6. Test conditions assume signal transition times of 5 ns or less, timing reference levels of 1.5V, input pulse levels of 0 to 3.0V, and output loading of the specified I_{OL}/I_{OH} and 30-pF load capacitance.
7. At any given temperature and voltage condition, t_{HZ} is less than t_{LZ} for all devices. Transition is measured ±500 mV from steady state voltage with specified loading in part (b) of AC Test Loads and Waveforms.
8. t_{HZCE} and t_{HZWE} are tested with C_L = 5 pF as in part (a) of Test Loads and Waveforms. Transition is measured ±500 mV from steady state voltage.
9. The internal write time of the memory is defined by the overlap of $\overline{CE}$ LOW and $\overline{WE}$ LOW. Both signal must be LOW to initiate a write and either signal can terminate a write by going high. The data input set-up and hold timing should be referenced to the rising edge of the signal that terminates the write.
10. $\overline{WE}$ is HIGH for read cycle.
11. Device is continuously selected. $\overline{CE} = V_{IL}$.

Switching Waveforms (continued)

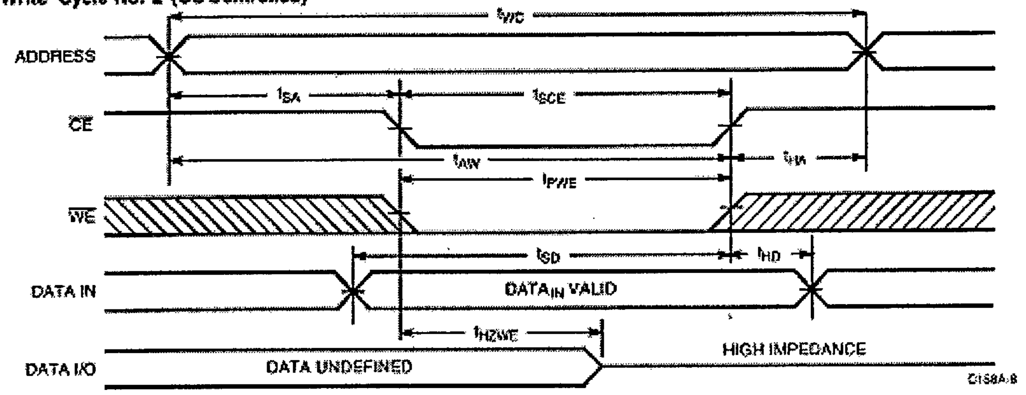
Read Cycle[10, 12]



Write Cycle No. 1 (WE Controlled)[9]

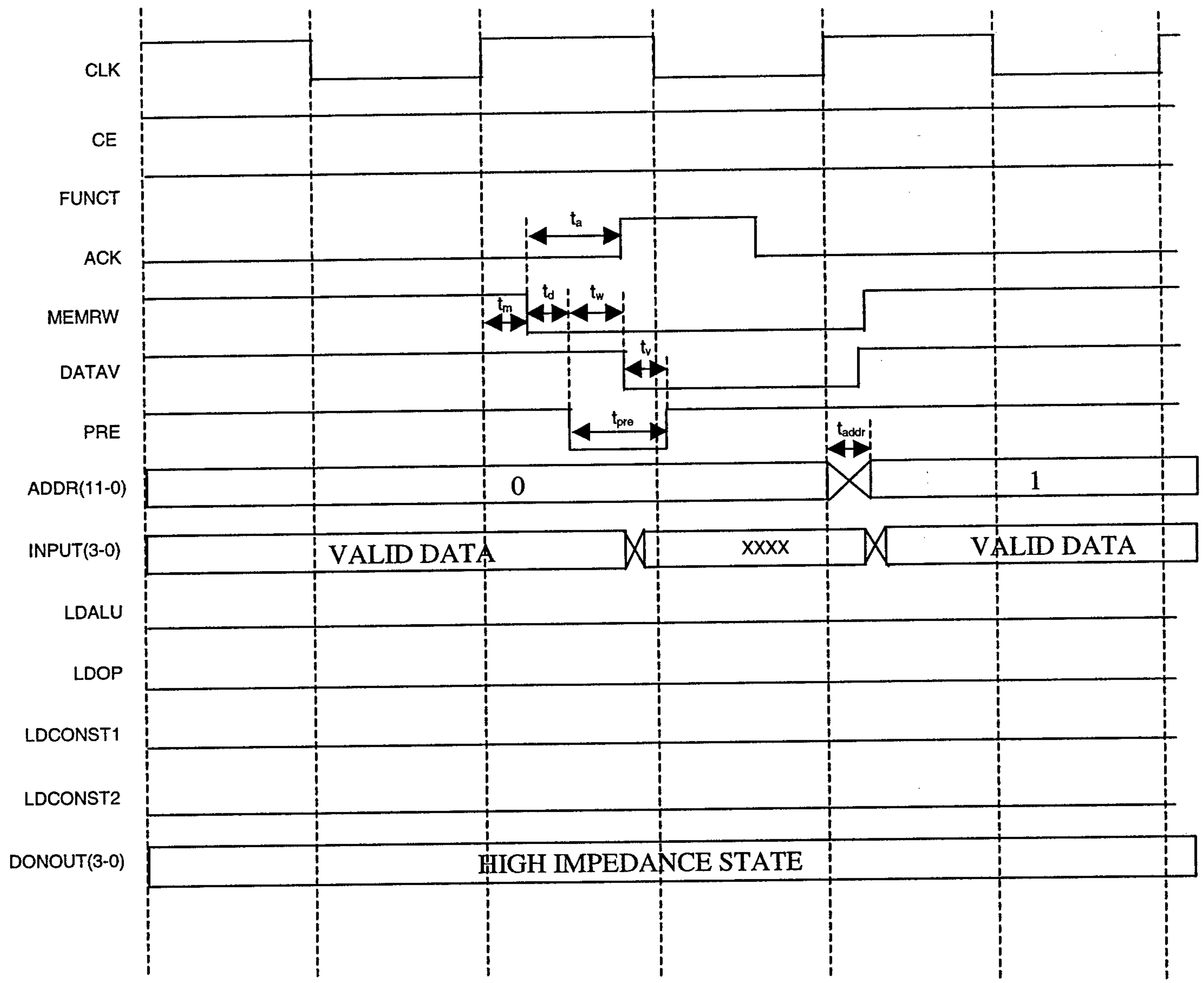


Write Cycle No. 2 (CS Controlled)[9, 13]



Notes:
12. Address valid prior to or coincident with CE transition LOW.
13. If CE goes HIGH simultaneously with WE HIGH, the output remains in a high-impedance state.

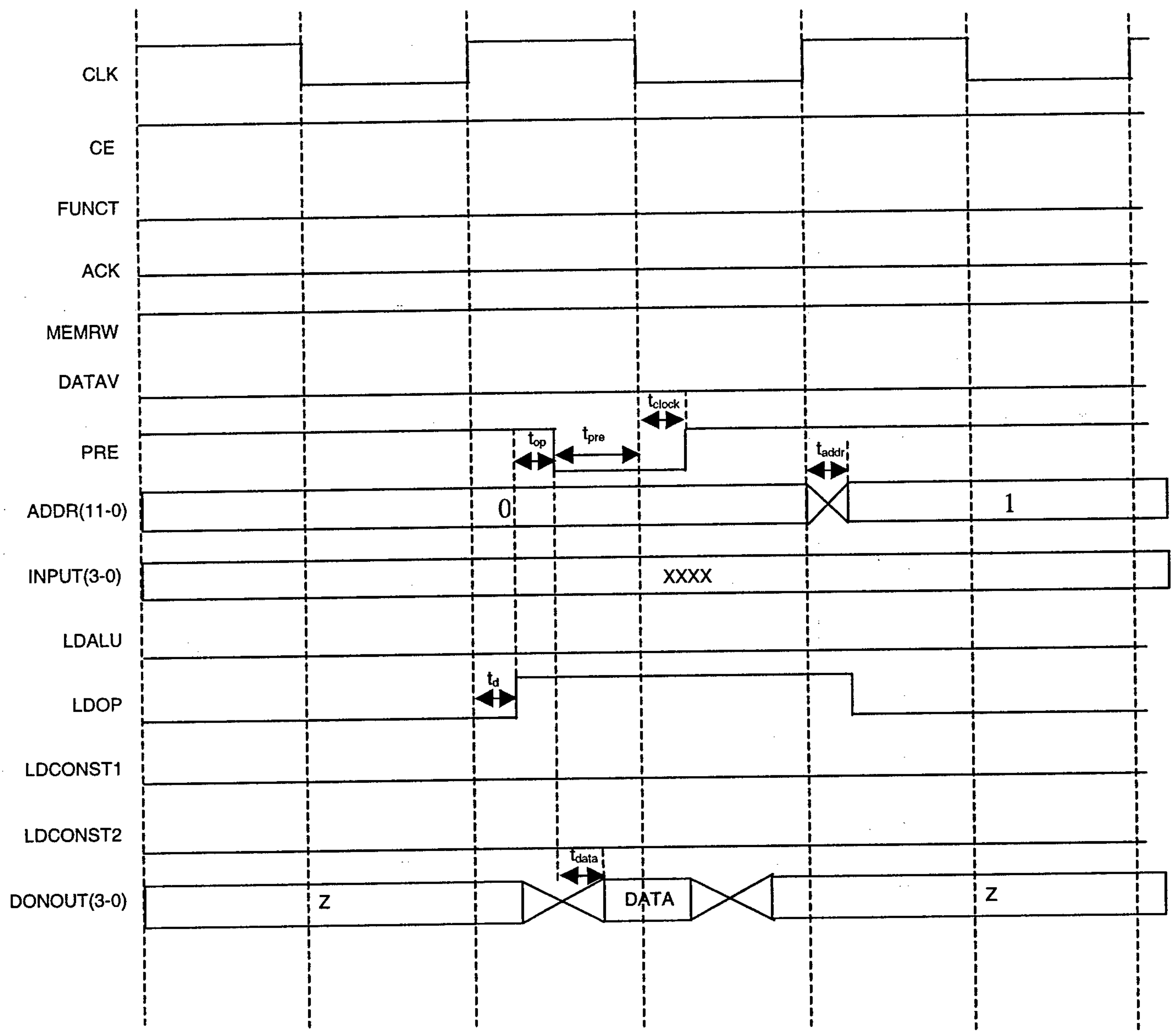
Program Memory Write Cycle



These timing delays take into account pad delays from MOSIS documentation of 0.8ns for inputs and 1.9ns for outputs. Signals coming from off-chip are assumed to have no delay from the off-chip device to the input pad

t_w = 5 ns
t_a = 8 ns
t_m = 2.8 ns
t_d = 2.4 ns
t_v = 3 ns
t_{pre} = 8 ns
t_{addr} = 1.2 ns

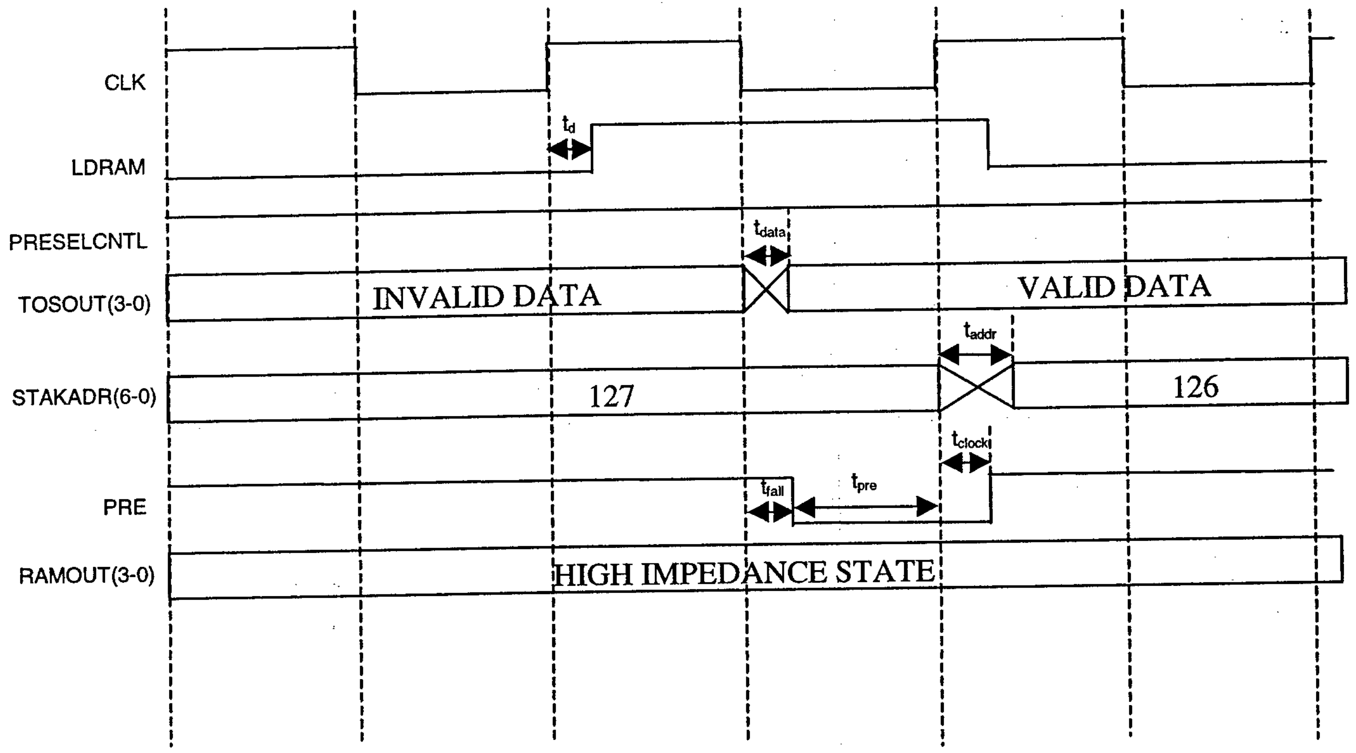
Program Memory Read



These timing delays take into account pad delays from MOSIS documentation of 0.8ns for inputs and 1.9ns for outputs. Signals coming from off-chip are assumed to have no delay from the off-chip device to the input pad

$t_d = 2.9$ ns
$t_{data} = 4$ ns
$t_{clock} = 2.8$ ns
$t_{op} = 3.5$ ns
$t_{addr} = 1.2$ ns
$t_{pre} = 4.5$ns

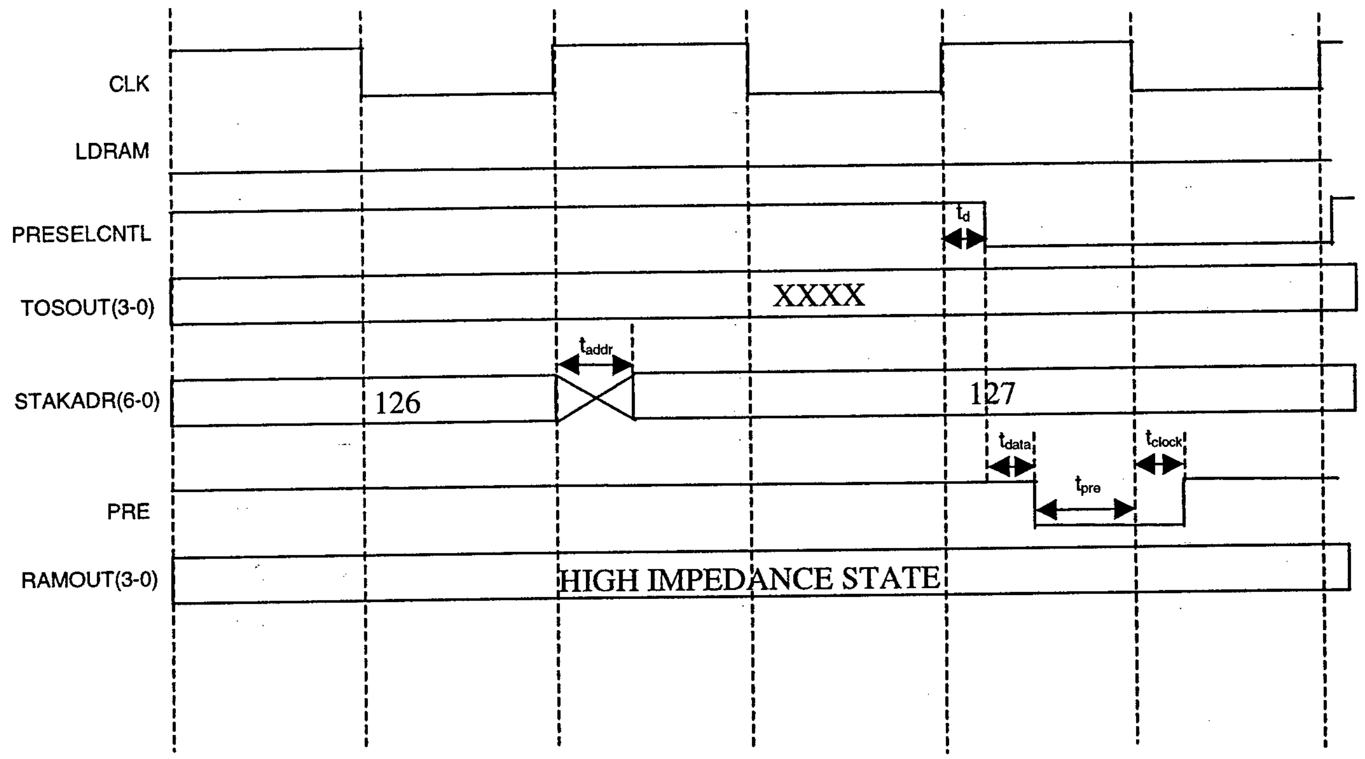
Stack Write



These timing delays take into account pad delays from MOSIS documentation of 0.8ns for inputs and 1.9ns for outputs. Signals coming from off-chip are assumed to have no delay from the off-chip device to the input pad

$t_d = 3.4$ ns
$t_{data} = 1.2$ ns
$t_{clock} = 1.2$ ns
$t_{fall} = 1.1$ ns
$t_{pre} = 3$ ns
$t_{addr} = 2.5$ ns

Stack Read



These timing delays take into account pad delays from MOSIS documentation of 0.8ns for inputs and 1.9ns for outputs. Signals coming from off-chip are assumed to have no delay from the off-chip device to the input pad

$t_d = 2.6$ ns
$t_{data} = 0.6$ ns
$t_{clock} = 1$ ns
$t_{pre} = 2$ ns
$t_{addr} = 2.5$ ns

Appendix D. OCTTOOLS Tutorial

Creating a Standard Cell Library from Radiation Hardened Cells or Commercial Cells

Introduction

The purpose of this tutorial is to learn how to place a standard cell library into the VLSI network for use with design tools. This tutorial will show how to use a different cell library than the Lager cell library throughout the entire design process. The finished product will be a layout using the different cells whether they are radiation hardened or another commercial set of library cells. The goal is to create .mag and .sdl files for each new library cell that will work with the Octtools. Octtools will lay out the circuit and perform the interconnections as described by the structural VHDL code.

*Note that you will need to have EENG653 and EENG695 Lab handouts for this tutorial assumes the student has taken both of these courses and performed these labs. This tutorial is targeted for VLSI thesis students who need to fabricate an integrated circuit with a different technology than the Hewlett-Packard 0.8 micron process using the Lager standard cell library.

*All commands that need to be performed at the Unix terminal prompt are specified by the following syntax (UnixPrompt% or HeraclesorEris%) before the actual command. Otherwise the commands are Magic commands and need to be carried out when Magic is open. For Magic commands it is crucial to have the cursor located within the Magic layout window, otherwise Magic will not recognize the command.

Preliminaries

First, you will need to copy files to your directory, which are needed to run through this tutorial. *All the different file types used throughout this tutorial are explained in the reference section of this tutorial located on the back page.* The .vhd and .sym files can be copied from (~kwatson/tutorial/), which includes a components.vhd file that instances all the individual cells. Also, files needed when using Octtools are located in this directory (lager, wolfe.rules_6.0, octprep, feed.mag, and all necessary net2sdlr files). The VHDL files are currently back-annotated with spice results from 0.5 sub-micron radiation-hardened standard cells. You can decide is this timing data is good enough to use when performing structural tests with your components. In addition, some sample .mag files (0_nan2.mag, 0_nan3.mag, 0_nan4.mag) that are Octtool complaint are in the subdirectory of the *tutorial* directory called *examples*. You can open these files in Magic and visually see how the standard library cells should be laid out at the end of standard library cell creation. Finally, a 0.35-micron standard library .cif file and a 0.50 micron radiation-hardened standard library .gds file in included in the *tutorial* directory.

From the directory that you want to work in for this tutorial, at the Unix prompt type:

UnixPrompt% cp –r ~kwatson/tutorial/* .

First, you need a different cell library than the Lager cell library preferably the cell library you will use for your design. However, I have provided two cell libraries for you to practice making a standard cell library. There are two different file types that a cell library can come in. First is the .cif format, which is the most popular format for commercial cell libraries. The second type is the .gds format, which is a common format for radiation hardened cell libraries. The MOSIS web site has different cell libraries for the different technology sizes and can be downloaded from *www.mosis.org*. However, the .cif files from MOSIS contain all the standard library cells in one .cif file for each technology. This creates a problem when trying to break apart the standard cells into individual Magic files. You may have to go into the main .cif file and copy sections out and paste them into separate .cif files. This can be easily done in EMACS by opening the main .cif file and opening a new file preferably with the nomenclature of cellname.cif. Once the standard cells are in individual .cif files a conversion from .cif to .mag files will have to be performed. My example .cif file (mtsms035dl.cif) is used for the TSMC 0.35 micron process, which has 4 layers of metal and 2 different layers of polysilicon.

To transform from .cif to .mag files simply open Magic with the correct technology called in the command at the Unix terminal prompt: (This is to call the technology files related to the TSMC 0.35-micron process)

UnixPrompt% magic –T SCN4M_SUBM.20.TSMC

Next use these commands in this order at the Magic terminal prompt to read in a .cif file and load the new Magic file into a Magic window:

:cif read filename

:open junk

:load filename

:save filename

Repeat this process until all the library cells you need are converted to .mag files and saved.

To transform from .gds to .mag files simply perform these commands:

I have provided an example .gds file (ga06lib6.gds) that is for the HP 0.5 micron process which has 3 layers of metal and 1 layer of polysilicon. This standard cell library is radiation hardened, so expect larger cell areas and irregular transistor dimensions especially for the n-channel transistors.
Resave the .gds files as .strm files at the Unix prompt

```
UnixPrompt% cp ga06lib6a.gds ga06lib6a.strm
```

Open Magic with the correct technology file loaded for the standard cells.

For example for HP 0.5-micron process at the Unix terminal prompt type:

```
UnixPrompt%  magic –T SCN3M_SUBM.30
```

At the Magic terminal perform the "calma" command to read in the .strm file into Magic:

```
:calma read ga06lib6a
```

Depending on how the library was made, you may have to call different cells into one window and then start performing the commands under the "***Transforming Magic Cells to Octtools Leaf Cells or Standard Cells.***" Cells possibly could have routing cells that will need to be loaded into the window with the standard cell main layout. Simply perform a "getc" command at the Magic prompt: (make sure the cursor is on the Magic Layout window you want the cell put into)

```
:getc filename
```

There are some standard files, which will make the conversion to a new standard cell library easier. First, behavioral VHDL code has been created for different standard cell names, so conforming to these names will make the process easier. In addition, .sym files exist for all these standard cell names which helps when working in Synopsys Graphical Environment.

The new standard cells should mimic the names listed below in the parentheses: (Keep in mind that the first character is the number zero, not the letter O)

(0_nan2)	2-input NAND gate
(0_nan3)	3-input NAND gate
(0_nan4)	4-input NAND gate
(0_nor2)	2-input NOR gate
(0_nor3)	3-input NOR gate
(0_nor4)	4-input NOR gate
(0_mux2)	2-input MUX
(0_xor2)	2-input XOR gate
(0_dff)	D Flip-Flop

(0_dffr)	D Flip-Flop w/Reset
(0_dffrnq)	D Flip-Flop w/Reset and without a negated output (not Q)
(0_aoi22)	And-Or-Invert (2 input AND gates and 2 input OR gate Structure)
(0_buf1x8)	Buffer (Increase drive strength by 8)
(0_buf1x1)	Buffer (Signal Refresh)
(0_zbuf1x8)	Tri-State Buffer (Increase drive strength by 8)
(0_zbuf1x1)	Tri-State Buffer (Signal Refresh)
(0_inv1x1)	Inverter
(0_inv1x8)	Inverter (Increase drive strength by 8)
(0_one)	Pull-Up (Makes Line a Constant Zero)
(0_zero)	Pull-Down (Makes Line a Constant One)

In order to view the .mag files in the correct technology, simply perform this command at the Unix terminal:

```
UnixPrompt%  magic –T SCN3M_SUBM.30 filename
```

Transforming Behavioral VHDL code to Structural VHDL code (including new library cells)

The transformation is time consuming, but simple to perform. Design Analyzer (DA) is a tool that reads in behavioral VHDL code and creates circuitry to perform the VHDL code's functions. *See EENG 695 Laboratory Exercise #2 in the Reference Section located in the back of this tutorial for help on how to transform behavioral VHDL code into a .db file.* It will use the Lager library cells to make the circuit. After optimizing the design and saving a .db file from DA, perform the "**db2sge**" command to transform the .db file into a readable format for the next tool (Synopsys Graphical Environment) to read.

```
UnixPrompt% db2sge –add_search_path ~cad/chiplib/synthesis –database filename.db
```

Synopsys Graphical Environment (SGE) will create the structural VHDL code from the following commands performed in SGE. *Make sure all the standard cell .sym and .vhd files are in the same directory you are creating your hierarchical components in*

Step 1: Open SGE

```
UnixPrompt%  sge &
```

Step 2: Select the **Schematic Editor** and **Double-Click on the Component** you want to create structural VHDL code for and check to make sure the design transferred over to SGE correctly.

Step 3: If the circuit has D flip-flops within it, then change out the Lager D flip-flops with the appropriate radiation-hardened D flip-flop or a commercial flip-flop and reconnect the appropriate wires. Reminder that some D flip-flops have a regular clock input along with a negated clock input, which may drive adding an additional inverter and buffer. If splicing the clock line to make the negated clock line, remember to put a buffer on the clock line after the splice point in order to make the delay similar in order for structural VHDL back-annotated timing tests function properly. For some reason without putting in the buffer into the clock line, the structural VHDL tests will yield U's on all the outputs. After performing your changes to the schematic window, perform a save by clicking on the **File** drop-down and click **Save.**

Step 4: Click-on the **Utility** drop-down menu on the main SGE window and click on **Schem to ASCII,** then scroll down and click on the component you wish to change out the Lager cells for your new standard cells. This will create a file with the syntax of component.asc. DO NOT CLOSE SGE.

Step 5: Open the component.asc file with EMACS and perform **Query Search** on the standard cell names. You will find the standard cell names towards the bottom of the .asc file, so go to the end of file and page up to the beginning of the standard cell declarations. After performing the modifications, click on the **File** drop-down menu and click **Save Buffer**.

Step 6: Click on the **Utility** drop-down menu on the main SGE window and click on **ASCII to Schem**. Scroll down and click on the component that you just changed in EMACS. Click on **Schematic Editor** on the side of the main SGE window and double click on the component you just read into SGE from the ASCII file. A window will open up showing the new schematic with the new standard library cells embedded into the top-level design.

Step 7: Select the **Tools** drop-down bar in the Schematic Editor window and then click on **VHDL Netlist**.

Step 8: Go into Structural VHDL Code and perform a **Query Search** in EMACS to the library directory and cell directory **Lager** to **WORK**. Make sure all the standard cell behavioral VHDL code is complied in your WORK directory before trying to compile the newly generated SGE structural VHDL code.

Creating .sdl Files from SGE for Octtools

Step 1: Open SGE

```
UnixPrompt% sge &
```

Step 2: Select **Navigate Hierarchy,** then click on **New**, click on the component you want to create the .sdl file for.

Step 3: Select **Process** in Editor Window and then click on **Netlist by Pin** in the drop-down menu

Step 4: Close the Editor window and save the file when prompted

Step 5: In main SGE window, click on **Symbol to ASCII** and select the same component as chosen in Step 2.

Step 6: Exit SGE, Click on **File** drop-down menu and select **Quit.**

Step 7: Open .net file in EMACS and perform a search and replace '/' with '_' globally. (After selecting **Query Search** and inputting the search character(s) and the replacement character(s), the Shift ! in EMACS will replace all the occurrences without having to type in "y" or "n" for each occurrence)

Step 8: Run **net2sdlr** on file (net2sdlr lets you work with the newly named standard cells rather than the Lager cell names) (may have to edit code if gates are missing in the .sdl file that is created)

Step 9: Open .sdl file in EMACS and check to be sure all gates were instanced.

Making Octtools Compliant Magic Cell Files

<u>Helpful Hints (keep these hints in mind as you are performing the next section):</u>

To create leaf cells from magic cells:

1. Make sure that Vdd! is the top rail and GND! is the bottom rail.

2. Cell width (metal rail width) must be a multiple of eight λ.
For example – cell width can be 8, 16, 24, 32, 40, 48, 56, ... etc.

3. Ensure that the lower left corner of GND! is at (0,3). You can use the black dot marker in the Magic window to locate the (0,0) point. You will need to use the **:move** command at the Magic terminal

4. Label Vdd! and GND! with vertical line labels along the border of all four Vdd! and GND! edges. See example .mag files to see what these labels look like.

5. Label all terminals in the Magic layout window. It is helpful to label the inputs/outputs as they are defined in the behavioral VHDL files. Center of terminal must be at x=m*8 + 4, where m in any positive integer or zero. All contacts must be vias labeled in the center. No terminals can have the same x value. Metal2 is used vertically to connect to the terminals. Any blockage may cause routing problems (can be blocked either upper or lower, but not

both).

To create labels for the input/output terminals make a box around the via and perform the label command:

:label labelname center (This will create a label that is centered on the via)

To check the x coordinate position; make a dot in the middle of the input/output vias and perform the box command (the xy coordinates will be displayed in the Magic terminal window):

:box

6. Create .SDL file from template below by replacing <cell_name>, <cell_width>, and <cell_height> with the actual parameters. Modify net statements and terminal statements as necessary. (An example is provided in the ~kwatson/tutorial/examples directory called **example.sdl**)

```
; sdl File
(parent-cell <cell_name> (FLAT_STOP "")
        (CELLAREA (* <cell_width> <cell_height>))
        (CELLCLASS LEAF))

(net A ((parent A1)))
(net B ((parent B1)))
(net C ((parent C2)))
(net D ((parent D2)))
(net O ((parent O)))
(terminal A1 (TERMTYPE SIGNAL) (DIRECTION INPUT))
(terminal B1 (TERMTYPE SIGNAL) (DIRECTION INPUT))
(terminal C2 (TERMTYPE SIGNAL) (DIRECTION INPUT))
(terminal D2 (TERMTYPE SIGNAL) (DIRECTION INPUT))
(terminal O (TERMTYPE SIGNAL) (DIRECTION OUTPUT))
(terminal Vdd! (TERMTYPE SUPPLY))
(terminal GND! (TERMTYPE SUPPLY))
(end-sdl)
```

7. Open a feed.mag cell. See example in Lager directory under the path (currently /cad/Lager5.0/Lager/common/LagerIV/cellib/stdcell2_3/misc) or the example provided from the initial copy from the tutorial directory.

8. Open the **lager** file and make sure the file looks like the syntax below. (This tells the Octtools to look for the .sdl files and the standard library cells in your current directory.)

```
(DMoct.sdl
        ./
)
(stdcell.leafcell
        ./
)
```

9. Make sure you have 'octprep' in your current directory. You can get a copy from (currently /cad/Lager5.0/Lager/common/LagerIV/cellib/stdcell/misc) or from the tutorial directory mention at the beginning.

10. Open wolfe.rules_6.0 file provided from the tutorial directory: (This file tells Octtools how it is going to route all your standard cells together- you can modify the rail lines extension from the main part of the component which is helpful for routing the power and round rail lines. For each technology size this file will have to be adjusted to get the right spacing between rows- this example file was used for the Hewlett-Packard 0.5 sub-micron process, a 3 layer metal, 1 layer of poly process.)

```
units_per_lambda      20
feedthru            ./feed physical
h_layer               MET1
v_layer               MET2
power_position        LEFT
v_net_weight          1.0
h_net_weight          1.0
rowSep                1.4
power_width           24
fast                  20
minimum_pad_space     50
restart               off
create_new_cel_file   off
vertical_wire_weight  1.0
vertical_path_weight  1.0
```

11. Remove any existing directories with the same name as the cells be created.

Transforming Magic Cells into Octtool Leaf Cells or Standard Cells:

Step 1: Load Complete Gate File into Magic

:load filename

Step 2: Perform Command at Magic Prompt
(use "test" for the filename to use as a dummy file)

:cif flat filename

Step 3: Quit Command

:q

Step 4: Open a Blank Magic File from the Unix Terminal

UnixPrompt% magic –T SCN3M_SUBM.30 junk

Step 5: Perform a Read of the New .cif File

:cif read filename

Step 6: Perform a Select in Magic – DO NOT EXPAND THE FILE

s

Step 7: Go into Edit Mode (be sure you have the cursor on the Magic window)

:edit

Step 8: Perform a Save on the Magic File

:save filename

Step 9: Perform a Load of the Magic File

:load filename

Step 10: Perform a Select on the Cell in Magic

s

Step 11: Measure the Area of the Cell in units of Lambda (λ)
(note to measure the cell with a box which encompasses the ground and Vdd rail lines-need to record this height and width in the .sdl file for each standard cell)

:box

Step 12: Rotate the Vdd (Power Line) on the top of the Cell

:clock

Step 13: Label According to .sdl names (inputs, outputs, Vdd, GND)
(may have to add vias for the inputs/outputs before putting on labels- see Magic Macros for the "pai via" macro)
(see the Helpful Hints section for Label Specifications, which are different for the inputs/outputs and the rail lines)
(see Lager cell .sdl/.mag pair for example of setup)
(need to orient cell so lower left corner is 3λ above (0,0))
(inputs and outputs will have to be separated by 4λ horizontally in the Magic layout)

Step 14: Box an Appropriate Area and Annotate in the .sdl file (include from bottom of GND! rail to the top of the Vdd! rail)

:box

Step 15: Perform a Save on the File

:save filename

Step 16: Quit Magic

:q

Step 17: Run Octprep (can do octprep *.mag at the end of adjusting all the files) (octprep calls two functions mag2oct and vulcan which are located at (~cad/Octtools5.0/sun4/bin/) make sure you have permissions to execute and read these binary files) (Before running Octprep you must be on a SUNOS station since they contain the Octtools software)

Make a Connection to where the Octtools software is located (switch over to either "eris" or "heracles" SUNOS stations).

UnixPrompt% rlogin eris
or
UnixPrompt% rlogin heracles

HeraclesorEris% octprep filename.mag
or
HeraclesorEris% octprep *.mag

Octprep creates a directory for each standard cell and contains files for the physical, structure_instance, and structure_master views that are needed by Octtools to create a Magic layout for a top-level component, which calls these standard cells.

Step 18: You now have a standard cell library after completing these steps for all the cells you need for your design and performing the "octprep" command on all of them.

Using Octtools to Generate Layout with New Library Cells

Step 1: Make sure all needed .sdl files, .mag files, and post-processed octprep directories for all the standard cells and higher level cells are in one directory. Be sure you are in that directory that contains all these files.

Step 2: Make a Connection to where the Octtools software is located (switch over to either "eris" or "heracles" SUNOS stations).

```
UnixPrompt%  rlogin eris
           or
UnixPrompt%  rlogin heracles
```

Step 3: Check to make sure you are in the directory mentioned in Step 1. (Initially, you will be in your root directory for example /home/newstudent2/kwatson/, may have to perform a couple of "cd" commands to get to the correct directory where all the .mag, and .sdl files are)

Step 4: Generate the Master View

```
HeraclesorEris% DMoct –m filename
```

Step 5: Generate the Structure_Instance View

```
HeraclesorEris% DMoct –s filename filename
```

Step 6: Perform a Schematic Check

```
HeraclesorEris% SIVcheck –s –m filename
```

Step 7: Perform an octprep on the .mag Files (This step should have already been performed, but just in case it hasn't it is a good idea to perform this command before the next DMoct command)

```
HeraclesorEris% octprep *.mag
```

Step 8: Generate the Layout (Dmoct will generate a .log file and place it in your current directory just in case any errors occurred)

```
HeraclesorEris% DMoct –#wolfe6_0 –t  ./wolfe.rules6_0# -#Stdcell –F# -v –l
filename filename
```

Step 9: Open the Magic file, which should be located in a subdirectory called (**layout**) and check to be sure everything is connected.

Comments when working with IRSIM and HSPICE files (Radiation Hardened Layouts Only):

(See EENG653 Laboratory Exercise #5 in the Reference Section for help with HSPICE and IRSIM file extraction from Magic) When trying to perform **ext2sim** or **ext2sp** on the extracted files from the final layout in Magic, remember to check on the transistor dimensions if using annular transistors for the n-channel transistors. For example, for the HP 0.5 micron process the .spice or .sim files have incorrect transistor dimensions when the files are generated. You will have to manually go in and change the transistor dimensions. For the 0.5 micron process, the dimensions were (1_51_) and they should be (_2_46_). You will need to go into the layout and count the lambda for the width and length measurements for the annular transistor. By performing, a **Query Search** in EMACS will take care of the transistor dimension problem.

Comments on using IRSIM and HSPICE with different technologies than the standard VLSI network 0.8-micron technology:

For IRSIM: You need the correct .prm file that goes with the technology you are trying to simulate. The .prm file can be in the directory you are running the simulation from, because by default IRSIM looks in your current directory for the .prm file and then looks at a specified location on the network for the file.

For HSPICE: MOSIS has test wafer run results with spice parameters included in the test run. It is best to take one of the most recent fabrication runs results and copy those spice parameters, which were deduced from testing the wafer. In your .run file simply use an (include) statement to call the spice parameter file that is copied from the wafer test results. It will take some searching through the wafer test run file to find the spice parameters, but you can find the default file on the VLSI network and compare what terms you need to what is in the wafer test run file.

Example 0_nan2.mag file with a few keynotes included:

```
magic
tech scmos
timestamp 913925037
<< metal1 >>
rect 0 3 56 19
rect 0 111 56 127
rect 0 111 56 127
rect 18 76 38 80
rect 26 34 30 76
rect 34 45 38 73
rect 18 45 22 73
rect 26 30 38 34
rect 18 19 22 27
rect 0 3 56 19
<< metal2 >>
rect 26 57 30 61
rect 18 57 22 61
rect 34 57 38 61
rect 34 57 38 61
rect 26 57 30 61
rect 18 57 22 61
<< poly >>
rect 39 73 41 119
rect 31 73 33 119
rect 23 73 25 119
rect 15 73 17 119
rect 31 71 41 73
rect 15 71 25 73
rect 34 69 38 71
rect 18 69 22 71
rect 34 45 38 49
rect 18 45 22 49
rect 35 37 37 45
rect 19 37 21 45
rect 31 35 41 37
rect 15 35 25 37
rect 39 22 41 35
rect 31 22 33 35
rect 23 22 25 35
rect 15 22 17 35
rect 31 20 41 22
rect 15 20 25 22
<< ppcontact >>  (Octtools changes psubstratepcontact to ppcontact- may
rect 46 5 50 9    have to add ppcontact to the .tech file so Magic can
rect 36 5 40 9    recognize this contact)
rect 26 5 30 9
rect 16 5 20 9
rect 6 5 10 9
<< pdcontact >>
rect 42 111 46 115
rect 26 111 30 115
rect 10 111 14 115
rect 34 76 38 80
rect 18 76 22 80
<< ndcontact >>
rect 34 30 38 34
rect 18 23 22 27
<< pdiff >>
rect 9 75 47 116
<< ppdiff >>
rect 0 51 56 55
rect 51 10 56 51
rect 0 10 5 51
rect 0 4 56 10
<< nndiff >>
rect 0 121 56 126
rect 51 67 56 121
rect 0 67 5 121
rect 0 63 56 67
<< m2contact >>
rect 34 57 38 61
rect 26 57 30 61
rect 18 57 22 61
<< ndiff >>
rect 9 14 47 43
<< pwell >>
rect -3 1 59 59
<< polycontact >>
rect 34 69 38 73
rect 18 69 22 73
rect 34 45 38 49
rect 18 45 22 49
<< nncontact >>    (Octtools changes nsubtratencontact to nncontact,
rect 46 121 50 125  may need to change the .tech file to recognize this
rect 36 121 40 125  new layer name)
rect 26 121 30 125
rect 16 121 20 125
rect 6 121 10 125
<< nwell >>
rect -3 59 59 129
<< labels >>
rlabel metal1 0 3 56 19 0 GND!    (Octtools will group the two Vdd! and
rlabel metal1 0 111 56 127 0 Vdd!  GND! terminals into one terminal
rlabel metal2 26 57 30 61 0 O1     each)
rlabel metal2 18 57 22 61 0 I1
rlabel metal2 34 57 38 61 0 I2
<< end >>
```

File Types

.vhd	Behavioral VHDL files used to describe the behavior of components and standard library cells
.sym	SGE files, which define the component's schematic representation as well as terminal- used by SGE to display a schematic view of component or standard library cell
.gds	Standard library cell file type that needs to be converted to .strm for Magic to read it- used to convert to .srtm files for Magic to read
.cif	Extraction view of a Magic layout- can be read by Magic with a :cif read command- used by Magic to create .mag files
.strm	File type used to be read by Magic with a :calma command- used by Magic to create .mag files
.sim	File type used by IRSIM for analysis purposes- used with IRSIM
.ext	Extraction file from Magic layout that is created by the Magic command :ext filename- a product produced from a .mag file using Magic
.mag	File type read by Magic that contains layout information- used by Magic to display layout

Appendix E. Design Checking Steps

There is a certain order of steps an engineer should go through before placing a chip onto an HP 82000 analyzer or any other kind of tester equipment. The steps are as follows:

Step 1: Check for continuity with a multimeter between power and ground pins with every other pin on the package. There should not be continuity between power or ground to any of the other pins, which are used for the output and input signals.

Step 2: Check for continuity between the power and ground pins. If there is continuity, then a short exists somewhere in the circuitry.

Step 3: If a short exist between power and ground, go back to the MAGIC file, which the design came from. Extraction of the top-level of the file in MAGIC will produce a file with a .ext extension. Perform an ext2spice on this file and open the .spice file that is created from this process. To check for power and ground problems, perform two searches in the .spice file. The first search should be for "GND pfet" in order to detect any p-channel transistors that are connected to ground. The last search is for "Vdd nfet" in order to detect any n-channel transistors that are connected to power.

Step 4: When finding wrongly connected transistors, use the bottom of the .spice file to reference back to which component contains the transistors. The final step is to go back into MAGIC and correct the errors found in the .spice file.

Step 5: Repeat this procedure until all of the transistors are connected correctly.

References

1. Donohoe, Greg W. *Microdot 4-bit Stack Machine: Architectural Description.* Unpublished. June 2000.

2. The MOSIS Service, Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Ray CA 90292-6695. http://www.mosis.org

3. Smith, Michael J. S. *Application-Specific Integrated Circuits.* Reading: Addison-Wesley, 1997.

4. Weste, Neil H. and Kamran Eshraghian, *Principles of CMOS VLSI Design A Systems Perspective, Second Edition.* New York: Addison Wesley, 1993.

5. IEEE Standard 1076-1993.

6. *Synopsys Design Analyzer Reference Manual,* Synopsys Incorporated, 700 East Middlefield Rd., Mountain View CA, 1998.

7. Shung, C. B, et al, "An Integrated CAD System for Algorithm-Specific IC Design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 10: pages 447-463, April 1991.

8. Barnhart, David J. *An Improved Asynchronous Implementation of a Fast Fourier Transform Architecture for Space Applications.* MS Thesis, AFIT/GE/ENG/99M-01. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1999.

9. Faggin, Federico, Hoff, Marcian E., Mazor, Stanley, and Shima, Masatoshi. *The History of the 4004.* IEEE Micro, Volume 16, pages 10-20, December 1996.

10. Betker, Michael R., Fernando, John S., and Whalen, Shaun P. *The History of the Microprocessor.* Bell Labs Technical Journal, Volume 2, No. 4, pages 29-56, Autumn 1997.

11. Donohoe, G. W., Lyke, J. C., Cannon, S. "Microdot: a Tiny Microcontroller for Distributed Sensor Interfacing." *Proc. 2nd International Conference on Integrated MicroNano Technology,* Pasadena, CA, 11-13 April, 1999.

12. Hollis, Ernest E. *Design of VLSI Gate Array ICs.* Englewood Cliffs, New Jersey: Prentice Hall, 1987.

13. Massara, R. E., editor. *Design & Test Techniques for VLSI & WSI Circuits.* London, United Kingdom: Peter Peregrinus Ltd., 1989.

14. Tanenbaum, Andrew S. *Distributed Operating Systems.* Upper Saddle River, New Jersey: Prentice Hall, 1995.

15. Holmes-Siedle, Andrew and Len Adams. *Handbook of Radiation Effects.* Oxford, New York: Oxford Press, 1994.

16. Brothers, Charles P., et al. "Radiation Hardening Techniques for Commerically Produced Microelectronics for Space Guidance and Control Applications." *20th Annual American Astronautical Society Guidance and Control Conference.* pages 169-180, February 1997.

17. The Materials Science and Engineering Department at Virginia Tech University. Online. Internet. 31 Jan 2001. Available: http://dvorak.mse.vt.edu/faculty/hendrick/mse4206/projects97/group02/hardening.htm

18. T. Ma and P. Dressendorfer. *Ionizing Radiation Effects in MOS Devices and Circuits.* New York: John Wiley and Sons, 1989.

19. Messenger, George C. and Milton Ash. *The Effects of Radiation on Electronic Systems, 2nd Edition.* New York: Van Nostrand Reinhold, 1992.

20. The National Institute for Nuclear Physics and High Energy Physics. Online. Internet. 31 Jan 2001. Available: http://www.nikhef.nl/user/dbello/txts/doctoral/adtol.html#geometrical

21. Osborn, J., Lacoe, R., Mayer, D., and Yabiku, G. "Total Dose Hardness of Three Commercial CMOS Microelectronics Foundries," *RADECS97 Proceedings.* pages 265-270, September 1997.

22. McDowell, Philip. *Choosing and using 4-bit microcontrollers.* New York: Marcel Dekker, Inc., 1993.

23. SanGregory, Sam L. *A Single-Chip 2K x 8-Bit Pipelined Digital RF Memory Using 2μm CMOS VLSI Technology.* MS Thesis, AFIT/GCE/ENG/92D-10. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1992.

24. Kranz, G. M. *The Design of a 6-Bit CMOS Dual-port Digital Radio Frequency Memory Using Very Large Scale Integrated Circuit Technology.* MS Thesis, AFIT/GE/ENG/90D-30, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1990.

25. MAGIC, Version 6.5.1, EECS/ERL Industrial Liaison Program, University of California at Berkley, Berkeley, CA.

26. IRSIM, Version 9.5, Stanford University, CA. 1988-1990.

27. HSPICE User's Manual Version 96.1, Meta-Software, Inc., 1300 White Oaks Road, Campbell CA, February 1996.

28. Using the HP82000 Manual, Hewlett-Packard, November 1993.

Vita

First Lieutenant Kirby M. Watson was raised in Princeton, Illinois. He graduated from Saint Charles High School in Saint Charles, Illinois in June 1993. He entered undergraduate studies at Marquette University in Milwaukee, Wisconsin where he graduated with a Bachelor of Science degree in Electrical Engineering in May 1997. He was commissioned through the Detachment 925A AFROTC at Marquette University where he was recognized as a Distinguished Graduate.

His first assignment was at Wright-Patterson AFB as a C-17 Maintenance Training Systems Engineer in June 1997. In August 1999, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the Air Force Technical Application Center (AFTAC) at Patrick AFB, Florida.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 07-03-2001	**2. REPORT TYPE** Mater's Thesis	**3. DATES COVERED** *(From - To)* May 00 - Mar 01

4. TITLE AND SUBTITLE MICRODOT- A 4-BIT SYNCHRONOUS MICROCONTROLLER FOR SPACE APPLICATIONS	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Kirby Michael Watson, First Lieutenant, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, 2950 P Street, WPAFB OH 45433-6583	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GE/ENG/01M-20
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/VSSE, Dr. Robert P. Pugh, 3550 Aberdeen Ave. SE, Kirtland AFB, NM 87117-5776, DSN 246-0585	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

13. SUPPLEMENTARY NOTES
Major Charles P. Brothers, Jr., PHD
DSN: 785-3636 ext. 4618, Email: Charles.Brothers@afit.edu

14. ABSTRACT
Satellites have limited power budgets due to the amount of power collected by the satellite's solar panels. The goal is to have a wide range of functionality, while running off a limited power source. Large microprocessors use large amounts of power to report back temperature and chemical sensor data to ground stations. By using small microcontrollers to perform the data collection and minimizing the usage of the larger microprocessors, the satellites will save power. A prototype design of the Microdot 4-bit microcontroller for space applications is presented. Requirements for the Microdot, such as microwatt power consumption and 23 different instructions, are based on research completed at AFRL/VSSE, Air Force Research Laboratory at Kirtland AFB, NM. A brief history of 4-bit microcontrollers and microprocessors, the synchronous design methodologies used, and space-based integrated circuit issues are presented. Various CAD tools were used, implementing both standard cell and full custom logic into the design. The prototype Microdot was fabricated at TSMC using MOSIS to validate the design implementation. Results from high fidelity simulations indicate the Microdot design has a power consumption of 16.3 mW operating at 1 kHz and consumes 22 mW when operating at the maximum operating clock frequency of 20 MHz.

15. SUBJECT TERMS
VLSI, synchronous, radiation hardened electronics, space electronics, microcontroller

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Major Charles P. Brothers, Jr., PHD
U	U	U	UU	171	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-3636 ext.4618